ALYSSON NEVES BESSANI

O PADRÃO UMIOP COMO BASE PARA COMUNICAÇÃO DE GRUPO CONFIÁVEL EM SISTEMAS DISTRIBUÍDOS DE LARGA ESCALA

FLORIANÓPOLIS 2002

UNIVERSIDADE FEDERAL DE SANTA CATARINA

CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

O PADRÃO UMIOP COMO BASE PARA COMUNICAÇÃO DE GRUPO CONFIÁVEL EM SISTEMAS DISTRIBUÍDOS DE LARGA ESCALA

Dissertação submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Mestre em Engenharia Elétrica.

ALYSSON NEVES BESSANI

Florianópolis, Setembro de 2002.

O PADRÃO UMIOP COMO BASE PARA COMUNICAÇÃO DE GRUPO CONFIÁVEL EM SISTEMAS DISTRIBUÍDOS DE LARGA ESCALA

Alysson Neves Bessani

'Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia Elétrica, Área de Concentração em *Automação e Sistemas*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.'

Prof. Joni da Silva Fraga, Dr.
Orientador

Prof. Edson Roberto De Pieri, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

Prof. Joni da Silva Fraga, Dr.
Presidente

Prof. Lau Cheuk Lung, Dr.
Co-Orientador

Prof. Frank Augusto Siqueira, PhD.

Prof. Mario Antonio Ribeiro Dantas, PhD

Prof. Rômulo Silva de Oliveira, Dr

Dedico este trabalho à magrinhagem que tem discernimento e não se afrouxa!

AGRADECIMENTOS

Primeiramente agradeço a Deus por me dar saúde e força para concluir este trabalho e a minha família (em especial minha mãe) por me apoiar durante todas as fases de sua concepção, mesmo estando longe.

Agradeço ao prof. Joni da Silva Fraga pela orientação e paciência, bem como ao Lau Cheuk Lung pela sugestão do tema e seus valiosos comentários durante o trabalho. Agradeço também ao Ricardo e a Luciana, bolsistas do LCMI, que tanto me ajudaram no início das implementações do MJACO e aos membros da banca por aceitarem o convite de participar dela.

Agradeço ao CNPq pelo apoio financeiro sem o qual não conseguiria realizar este trabalho.

Agradeço aos meus amigos/sócios da *Rendera Soluções em Comunição Empresarial* (Flávio, Flávio e Marcel) por aguentarem as pontas enquanto eu estava em fbripa "aprendendo a surfar".

Agradeço aos amigos de Maringá que reencontrei em Florianópolis - Cassia (valeu a ajuda pelo abstract, os erros que vão aparecer são frutos de minhas alterações), Dani, Rento e Alessandro - e os que fiz em aqui pelos bons momentos, em especial ao pessoal do LCMI - Tércio, Emerson, Ricardo, Luis Fernando (paquito), Carlos, Tomas, Frank, Luciano, Fabio, Fabio (Pinga), Priscila, Patricia, Ana, e tantos outros - que me receberam muito bem nesta terra. Agradeço também ao pessoal da computação (e afins) pelos churrascos de nível e as festas com peso realizadas: Balzan, Thirzá, Renato, Guto, Dione, Ricardo, Michael, Junior, Lidiane, Renata, André, Guilherme e alguns outros parceiros eventuais.

Um agradecimento especial aos grandes amigos que tiveram o (des)prazer de morar comigo: Fernando Barreto, que me recebeu quando eu cheguei e não conhecia nada; Beto, que me aguentou por 4 anos na graduação e ainda morou comigo mais um ano aqui; Adamo e Rafael, meus convivas atuais.

Finalmente gostaria de agradecer à, hoje falida, fábrica das tubainas ouro verde (a verdadeira tubaina), ao inventor anônimo da paçoca rolha (humm, que delícia), ao reponsável pela fabricação da cachaça Ypioca (sucesso!), ao mestre cervejeiro responsável pela Skol (a única que não da dor de cabeça), à *Mirabilis* pela invenção do ICQ e quero agradecer mais uma vez a Deus, só que agora pela invenção da mulher.

E antes que eu me esqueça quero declarar: Elvis é o rei (mas eventualmente o magnífico Roberto Carlos pode também ser considerado rei)!

Resumo da Dissertação apresentada à UFSC como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia Elétrica.

O PADRÃO UMIOP COMO BASE PARA COMUNICAÇÃO DE GRUPO CONFIÁVEL EM SISTEMAS DISTRIBUÍDOS DE LARGA ESCALA

Alysson Neves Bessani

Setembro/2002

Orientador: Prof. Joni da Silva Fraga, Dr.
Co-Orientador: Prof. Lau Cheuk Lung, Dr.
Área de Concentração: Automação e Sistemas
Palavras-chave: CORBA, Comunicação de Grupo, Difusão Confiável, Tolerância a Faltas
Número de Páginas: xiii + 82

O conceito de grupo de objetos é uma abstração largamente usada no tratamento de replicação em sistemas distribuídos tolerantes a faltas. Os sistemas que se baseiam neste conceito geralmente utilizam algum tipo de suporte de comunicação de grupo (SCG), que oferece comunicação multiponto através de primitivas com níveis diferenciados de confiabilidade de entrega e ordenação de mensagens. Em sistemas de larga escala, como a Internet, o *multicast* IP é a tecnologia base para a construção de protocolos de difusão seletiva confiável, que se constituem na base dos SCG. A junção dos conceitos de objetos distribuídos e de grupo em suportes de *middleware*, em especial os que seguem a arquitetura CORBA, tem sido objeto de pesquisas recentes, que motivaram a OMG a lançar duas especificações no sentido de introduzir o conceito de grupo nesta arquitetura: FT-CORBA (*Fault-Tolerant CORBA*), que introduz alguns serviços para suporte à tolerância a faltas nesta arquitetura, e o UMIOP (*Unreliable Multicast Inter-ORB Protocol*), que provê um protocolo de difusão não confiável (MIOP) baseado em *multicast* IP e um modelo de objetos para suportar este protocolo em ORBs CORBA. Com estes dois padrões a OMG separa claramente aspectos de comunicação de grupo (UMIOP) e de *membership* (FT-CORBA). Este trabalho propõe um modelo de integração entre o FT-CORBA e o UMIOP utilizando o ReMIOP (*Reliable* MIOP), que corresponde ao protocolo MIOP acrescido de controle de fluxo e retransmissões a fim de dar confiabilidade a este, como base para suporte de comunicação de grupo. A concretização destas experiências com comunicação de grupo no CORBA deram origem ao MJACO, um ORB que implementa o padrão UMIOP e suas extensões para confiabilidade, bem como mantém suas capacidades de comunicação ponto-a-ponto. A existência do MJACO abre espaço para a implementação do modelo proposto e uma série de pesquisas sobre *middleware* adaptativo.

Abstract of Dissertation presented to UFSC as a partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

THE UMIOP STANDARD AS A BASE FOR RELIABLE GROUP COMMUNICATION IN LARGE SCALE DISTRIBUTED SYSTEMS

Alysson Neves Bessani

September/2002

Advisor: Prof. Joni da Silva Fraga, Dr.
Co-Advisor: Prof. Lau Cheuk Lung, Dr.
Area of Concentration: Automation and Systems
Key words: CORBA, Group Communication, Reliable Multicast, Fault Tolerance
Number of Pages: xiii + 82

The concept of object group is a largely used abstraction when treating replications in fault-tolerant distributed systems. The systems based in the group concept often use some sort of group communication support (GCS), which offers multipoint communication through a series of primitives with differentiated levels of messages reliability and delivery ordering. In large scale systems, as the Internet, the multicast IP is the basis technology for the construction of reliable multicast protocols, which compose the GCS basis. The union of distributed objects and group concepts in middleware support, specially the ones which follows CORBA architecture, have been object of recent research, that motivated OMG to launch two specifications towards introducing the group concept in this architecture: FT-CORBA (Fault-Tolerant CORBA), which introduce a series of services to support fault-tolerance in this architecture, and the UMIOP (Unreliable Multicast Inter-ORB Protocol), which provides an unreliable multicast protocol (MIOP) based in multicast IP and an object model to support this protocol in CORBA ORBs. With these two standards the OMG clearly separates group communication (UMIOP) and membership (FT-CORBA) aspects. The UMIOP can be used as a group communication support for FT-CORBA, once this important component is expected but not specified, since it offers some warranties. This work proposes an integration model for the FT-CORBA and the UMIOP using the ReMIOP (Reliable MIOP), which corresponds to the MIOP protocol plus flow control and retransmission, to provide reliability. The process of making these experiences with group communication in CORBA concrete, originated the MJACO, an ORB that implements the UMIOP standard and its extensions for reliability, as well as maintains its poit-to-point communication capabilities. The existence of MJACO provides means to implement the proposed model and may support many researches about adaptative middleware.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

A arquitetura CORBA (*Common Object Request Broker Architecture*) [21] introduzida pela OMG (*Object Management Group*), corresponde nos dias de hoje as mais bem sucedidas especificações de *middleware* para suporte a sistemas de objetos distribuídos. O principal componente desta arquitetura é o ORB (*Object Request Broker*) que entre outras coisas implementa as semânticas de comunicação definidas na arquitetura. As mensagens geradas nas comunicações seguem uma sintaxe de transferência própria: o *General Inter-ORB Protocol* (GIOP). Esta sintaxe torna as mensagens envolvidas nas comunicações independentes das implementações de ORBs e das conseqüências de um ambiente heterogêneo. O mapeamento do GIOP sobre o TCP/IP é concretizado através do IIOP, *Internet Inter-ORB Protocol*.

O IIOP, seguindo as características do TCP, foi definido para comunicações ponto a ponto. Apesar da combinação IIOP - TCP/IP corresponder a uma boa solução para comunicações que seguem este modelo - abordando aspectos como controle de erro, ordenação de mensagens, etc. - muitos outros paradigmas de comunicação quando implementados sobre estes protocolos não aproveitam as características dos níveis mais baixos da rede. O reflexo desta dificuldade sempre recai em custos no desempenho destas comunicações.

Muitas aplicações distribuídas dependem de abstrações como grupos de objetos ou da necessidade da disseminação de dados entre vários *hosts* da rede. Estas aplicações geralmente utilizam-se de um suporte de comunicação de grupo que ofereça serviços de comunicação multiponto com algumas propriedades como garantias de entrega e ordenação, bem como desempenho adequado. Como um primeiro passo para introduzir um suporte a comunicação de grupo na arquitetura CORBA, a OMG publicou em 1999 um RFP (*Request For Proposal*) onde definia uma série de requisitos para um serviço de difusão não confiável baseado em *multicast* IP. O *multicast* IP compreende um conjunto de extensões ao protocolo IP que o habilita na concretização de comunicações multiponto [13]. Este protocolo se caracteriza pela ausência de garantias e pelo alto desempenho, especialmente em redes locais.

Em 2001, como resposta à RFP, duas propostas foram submetidas a OMG: a primeira vinha da *Inprise* (antiga *Borland*) e a outra foi desenvolvida por um conjunto de empresas e instituições de pesquisa, dentre as quais destacam-se a *Eternal Systems*, *IONA*, *Object Oriented Concepts* e a *University of California, Santa Barbara*. Esta segunda proposta foi a vencedora e se tornou a base das especificações homologadas pela OMG no final de 2001. Como conseqüência destes esforços de padronização, surgiu então o MIOP (*Multicast Inter-ORB Protocol*), protocolo responsável pelo mapeamento do GIOP sobre a pilha UDP/*multicast* IP. O MIOP é o elemento chave para tornar disponível um serviço de difusão não confiável em suportes CORBA de objetos distribuídos.

A introdução do paradigma de grupo em padrões abertos tem sido alvo de vários trabalhos de pesquisa e propostas de padronização [1, 14, 22, 34, 37, 46]. Prover suporte de grupo a aplicações distribuídas envolve uma combinação de protocolos que tratam do gerenciamento de grupo (*membership*), detecção de falhas, transferência de estado e comunicação de grupo. Dentro da OMG, essas abstrações estão sendo padronizadas separadamente, os três primeiros são tratados nas especificações FT-CORBA [20]. Todavia, a OMG ainda não tem publicado uma especificação para a comunicação de grupo na arquitetura CORBA que atenda, por exemplo, diferentes níveis de garantias nas várias versões possíveis deste paradigma. O MIOP e seu modelo de objetos, definidos pela especificação UMIOP (*Unreliable Multicast Inter-ORB Protocol*)[1] [22], são um primeiro passo na definição deste tipo de suporte, visto que, a partir deste padrão, é possível construir serviços mais elaborados.

Este trabalho apresenta um estudo a respeito da integração de um serviço de difusão não confiável em um *middleware* CORBA, sua integração com outra especificação que trata de grupos (o FT-CORBA), e a extensão deste suporte para oferecer serviços de difusão confiável.

1.1 Motivação e Objetivos

Nos últimos anos, vários trabalhos envolvendo grupo em suportes de objetos distribuídos foram realizados, resultando em protótipos ou mesmo produtos que mostram a viabilidade da junção de processamento de grupo a estes suportes sem que estes percam suas características de sistemas abertos. Todo este esforço motivou a OMG para se lançar na padronização de um serviço de difusão não confiável, que deve ser o primeiro passo para um serviço mais confiável de comunicação de grupo dentro do CORBA.

Assim, este trabalho vem no sentido de avaliar as recentes especificações da OMG que introduzem um suporte a comunicação de grupo bastante simples, e propor extensões a este suporte, a fim de torna-lo mais adequado à implementação de serviços tolerantes a falhas.

Os objetivos principais deste trabalho são:

[1] UMIOP corresponde as especificações do protocolo MIOP e seu modelo de objetos.

- Estudar o padrão UMIOP, proposto pela OMG, e avaliar aspectos de integração deste padrão a ORBs existentes;
- Implementar as especificações UMIOP em um ORB de código aberto;
- Estudar o padrão FT-CORBA e as possibilidades da integração do UMIOP como suporte de comunicação de grupo deste;
- Estudar a literatura de protocolos de difusão confiável e propor extensões ao protocolo da OMG visando adicionar propriedades de confiabilidade a este;
- Implementar um ORB, que permita comunicação ponto a ponto, difusão não confiável e difusão confiável, tudo de maneira padronizada. Este ORB deve servir de suporte de comunicação de grupo para o FT-CORBA.

1.2 Estrutura da Dissertação

O capítulo 2 desta dissertação apresenta uma introdução aos principais conceitos envolvidos na especificação de suportes a comunicação de grupo. São tratados os parâmetros que definem um sistema distribuído, e como conseqüência, o suporte a comunicação de grupo deste, e os vários níveis de garantias que podem ser oferecidos por este suporte através de diferentes primitivas de comunicação.

O capítulo 3 apresenta aspectos da comunicação de grupo em sistemas de larga escala. Neste capítulo o *multicast* IP é apresentado como suporte básico de comunicação multiponto na Internet e como base para a construção de uma série de tipos de protocolos de difusão confiáveis para redes de larga escala.

O capítulo 4 apresenta a arquitetura CORBA e os esforços da OMG para introduzir o conceito de grupo nesta arquitetura. Neste capítulo serão apresentados os padrões FT-CORBA e UMIOP. Ao final do capítulo é proposto um modelo de integração entre estes dois padrões, onde o UMIOP, acrescido de algum mecanismo que garanta confiabilidade é utilizado como suporte de comunicação de grupo do FT-CORBA.

O capítulo 5 apresenta o ReMIOP (*Reliable* MIOP), um conjunto de extensões ao protocolo MIOP para que este ofereça serviços de difusão confiável. Os algoritmos para o ReMIOP são descritos formalmente neste capítulo.

O capítulo 6 apresenta a implementações do MJACO - um ORB com suporte a serviços de difusão não confiável (através do MIOP) e confiável (através do ReMIOP). Um pequeno conjunto de testes de desempenho e escalabilidade também é apresentado neste capítulo.

Finalmente, o capítulo 8 apresenta as conclusões deste trabalho e possíveis extensões que podem ser abordadas em trabalhos futuros.

Capítulo 2

Conceitos de Comunicação de Grupo

Grande parte dos sistemas distribuídos existentes atualmente trabalha dentro do paradigma cliente/servidor, onde alguns componentes tem o papel de servidor (provendo um ou mais serviços) enquanto outros de cliente (utilizando-se desses serviços). Os principais mecanismos de comunicação usados neste modelo são o RPC (*Remote Procedure Call*) e o RMI (*Remote Method Invocation*). Estes mecanismos encapsulam as comunicações entre processos, que ocorrem geralmente através de troca de mensagens, em chamadas de procedimento (RPC) e de métodos a objetos remotos (RMI) e se caracterizam por serem bloqueantes (para o cliente), ponto-a-ponto, e assimétricos. Entretanto, algumas aplicações necessitam de um serviço de comunicação com características não bloqueante, multi-ponto e simétrico. Estas aplicações geralmente utilizam-se de mecanismos ou suportes de comunicação de grupo (SCG) para atender suas necessidades.

Um suporte de grupo ou de comunicação de grupo implementa um conjunto de abstrações e funcionalidades que permitem a associação de objetos ou processos como receptores de comunicação um para muitos. A associação de processos ou objetos em uma relação de grupo não necessariamente defi ne uma replicação de código e dados. Esta associação pode caracterizar objetos e processos diferentes, relacionados em grupo por questão de otimização na distribuição de informações. No modelo de comunicação de grupo, os atores e alvos das atividades de processamento podem se comunicar em um sentido mais nivelado no estilo *peer-to-peer*, ou mesmo, implementar versões de mecanismos mais convencionais como RPC e RMI para grupo. Em todas estas alternativas de interfaces, o mecanismo básico usado nas comunicações é a difusão seletiva, que em suas diferentes versões implementa propriedades com graus diferenciados de confi abilidade e ordenação.

O conceito de grupo, no contexto de sistemas distribuídos, é geralmente utilizado para minimizar a complexidade das grandes aplicações ou para prover aspectos não-funcionais em sistemas distribuídos. Um grupo, de uma maneira menos restritiva, é uma coleção de objetos, ou processos, referenciados por um único nome ou endereço [40]. Estes processos trabalham de maneira coordenada para atingir um objetivo, como o compartilhamento de carga de processamento (distribuindo

requisições através dos membros do grupo), prover serviços tolerantes a faltas (se um processo do grupo falha, outro membro assume seu lugar) ou mesmo distribuir conteúdo multimídia pela internet.

Existem vários sistemas de comunicação de grupo descritos na literatura [16, 35, 38, 41, 46, 47], estes sistemas geralmente trabalham com grupos de processos comunicantes através de um mecanismo de IPC (*Inter-Process Communication*). Entratento, vários *middlewares* que implementam objetos distribuídos incluem entre suas funcionalidades o suporte a comunicação de grupo. O JavaRMI e o CORBA estão entre estes sistemas, já que implementações que suportam grupos já estão disponíveis, como por exemplo o Jgroup [36] para JavaRMI e o GroupPac [31] para o CORBA. Estas implementações geralmente utilizam algum sistema de comunicação de grupo baseado em troca de mensagens em um nível inferior, construindo seu mecanismo de RMI sobre este sistema, estando sujeitas portanto as semânticas associadas a este último.

Este capítulo apresenta uma breve introdução ao modelo de comunicação de grupo. Inicialmente serão apresentados os parâmetros que caracterizam os sistemas distribuídos onde serão usados os modelos de comunicação de grupo. Em seguida alguns conceitos e componentes básicos destes suportes são discutidos, e finalmente os diversos tipos de primitivas de comunicação de grupo, definidas nestes suportes, serão apresentadas.

2.1 Ambiente de Computação Distribuída

Os modelos de ambientes computacionais onde a comunicação de grupo está inserida, com suas características, define os serviços e algoritmos de difusão de mensagens em suas propriedades e nas suas limitações. Por modelo ou ambiente computacional, identificamos, entre outras coisas, as hipóteses de falhas às quais os algoritmos e serviços devem sobrevir e as hipóteses de sincronismo do sistema com os seus limites ou não nas suas latências. Na sequência são apresentados os principais pontos que definem estes modelos computacionais distribuídos.

2.1.1 Sincronismo

O sincronismo é um atributo de modelos computacionais e sistemas distribuídos que está relacionado ao comportamento tanto dos processos quanto da própria rede. É este atributo que define os limites de tempo para o processamento de determinados eventos relacionados à computação distribuída.

Modelos ou sistemas síncronos são caracterizados basicamente por limites em latências de comunicação e em prazos de processamento[1]. Assim, um sistema é considerado **síncrono** se ele satisfaz as seguintes propriedades [23]:

[1]Os prazos em processamento podem ser verificados quando se assume a noção de passos de processamento. Se con-

1. Cada processo p pertencente ao sistema tem um relógio local C_p que, dada a constante positiva ρ ($\rho << 1$), apresenta um desvio limitado em relação ao padrão de tempo, tal que:

$$(1-\rho) \leq \frac{C_p(t) - C_p(t')}{t - t'} \leq (1+\rho) \tag{2.1}$$

   onde $C_p(t)$ e $C_p(t')$ são as leituras de C_p nos pontos de tempo real t e t';

2. Existe um limite superior para o tempo de atraso de uma mensagem. O tempo de atraso consiste no tempo necessário para enviar, transportar e receber uma mensagem através da rede (latência de mensagem);

3. Existe um limite superior para o tempo necessário para que um processo execute um passo de processamento.

A grande vantagem de se ter um sistema síncrono é que neles é possível estabelecer restrições temporais facilmente, permitindo, por exemplo, a implementação de detectores de falhas perfeitos [9]. Na prática, a construção deste tipo de sistema, quando possível, envolve uma certa complexidade, o que, muitas vezes, impossibilita sua construção. Na Internet, por exemplo, que é uma rede de larga escala compartilhada por milhões de usuarios, é impossível encontrar limites máximos ou mínimos para o atraso no envio de mensagens, pois a carga da rede no momento do envio e o caminho que os pacotes contendo os dados da mensagem podem tomar até seu destino são praticamente imprevisíveis.

Enquanto numa ponta do espectro de sincronismo temos o modelo síncrono, onde todas as variáveis importantes do ambiente são limitadas e conhecidas (um modelo extremamente favorável à implementação de SCG), na outra ponta temos o modelo **assíncrono**, onde não existe nenhum tipo de limitação para as variáveis do ambiente computacional (um modelo muito desfavorável à implementação de SCG).

No modelo assíncrono não existem limites para o atraso das mensagens, o desvio dos relógios (geralmente não se assume a disponibilidade de relógios no modelo) e nem para o tempo de execução de passos de processamento em processos. Em suma, considerar um sistema assíncrono significa não assumir nenhum limite de tempo no sistema. Este modelo é muito interessante se considerarmos sua semântica simples e sua capacidade de refletir sistemas com carga inesperada (como a Internet).

Os modelos síncronos e assíncronos de computação são apenas dois extremos de um amplo espectro de modelos de sincronismo. Modelos intermediários relaxam uma ou mais características dos sistemas síncronos, como por exemplo, o limite superior para os passos de execução de um processo, fazendo com que estes se aproximem do modelo assíncrono.

siderarmos que a execução de um processo é caracterizada por um autômato então cada passo de processamento é uma transição neste autômato.

Dois modelos de sincronismo parcial são especialmente interessantes e tem sido alvo de intensas pesquisas: modelo de tempo livre (*time-free*) [17] e o assíncrono temporizado (*timed asynchronous*)[12].

O modelo de tempo livre é utilizado como base na maioria das pesquisas em sistemas assíncronos. Este modelo é definido pelas seguintes propriedades:

1. Não existem quaisquer limites de tempo para a execução dos serviço do sistema;
2. A comunicação entre dois processos é confiável;
3. Os processos só sofrem falhas de parada (ver próxima sub-seção);
4. Os processos não tem acesso à relógios de hardware.

Este modelo é bastante utilizado devido à sua semelhança com a Internet. Por exemplo, a propriedade 2 descreve exatamente o comportamento de uma conexão TCP. Entretanto, a maioria dos serviços de interesse na construção de SCG, como consenso, eleição de líder [17] e *membership* [8] não podem ser implementados neste tipo de sistema.

O modelo assíncrono temporizado define uma série de propriedades que visam identificar comportamentos síncronos durante execuções assíncronas. Para isto são definidas algumas premissas que precisam ser sustentadas neste modelo:

1. Todos os serviços são temporizados: todos os serviço no sistema tem em sua especificação um limite de tempo para gerar saídas em resposta a entradas;
2. A comunicação entre processos é feita através de um serviço de datagrama não confiável sujeito a falhas de omissão e desempenho;
3. Os processos sofrem falhas de parada e desempenho (ver próxima sub-seção);
4. Os processos tem acesso à relógios de hardware com erro linear (de acordo com a equação 2.1);
5. Não existem limites no número de falhas de comunicação e de processos no sistema.

Este modelo de sincronismo também pode ser visto como uma abstração que descreve, com bastante precisão, as redes e sistemas distribuídos disponíveis atualmente. Já que hoje em dia, qualquer computador está equipado com um relógio de alta precisão e serviços de datagrama não confiáveis são largamente disponíveis através de protocolos como o UDP [39]. Os serviços temporizados, definidos pela premissa 1, também são factíveis, visto todo serviço tem o seu *time-out* associado, mesmo que seja por parte do usuário (cancelando a requisição do serviço).

No modelo assíncrono temporizado, ao contrário do modelo de tempo livre, praticamente todos os serviços de interesse em SCG podem ser implementados [12].

Além destes modelos que flexibilizam a abordagem síncrona para obter aproximações de modelos assíncronos, existem outras abordagens para modelos assíncronos. São os casos de algoritmos probabilístas [4] e de detectores de falha não confiáveis [9].

2.1.2 Tipos de Faltas nos Processos

Um processo é chamado **falho** se durante sua execução seu comportamento torna-se diferente do especificado para o algoritmo que ele executa. Os vários tipos de desvios das especificações que um processo pode experimentar definem os tipos de falhas parciais a que o sistema está sujeito. A lista a seguir apresenta os principais tipos de falhas de processo encontrados na literatura [23]:

- **Parada:** O processo faltoso para de funcionar prematuramente. Este é o tipo de falha mais simples, e ocorre, por exemplo, quando a máquina em que o processo está executando é desligada;
- **Omissão de Envio:** O processo faltoso se omite de maneira aleatória ou eventual de enviar as mensagens que eram esperadas dele;
- **Omissão de Resposta:** O processo faltoso não recebe de maneira aleatória ou eventual as mensagens enviadas à êle;
- **Arbitrária:** O processo faltoso pode exibir qualquer comportamento. Pode enviar mensagens inesperadas ou mudar de estados arbitrariamente. Este tipo de falta também pode ser chamado de **maliciosa** ou **bizantina**;
- **Arbitrária com Autenticação:** Igual a anterior, porém um mecanismo de autenticação por chaves não forjaveis está disponível no sistema, permitindo a identificaç ão correta do comportamento bizantino (detecção viável).

Estes tipos de faltas valem tanto para sistemas síncronos quanto para sistemas assíncronos, e podem ser classificados de acordo com sua severidade. A figura 2.1 apresenta uma classificação das falhas apresentadas nesta seção de acordo com sua severidade.

Na figura 2.1 as setas representam o sentido de um aumento de severidade da faltas. Por exemplo, As faltas arbitrárias são mais severas que as faltas arbitrárias com autenticação pois nas primeiras um processo faltoso pode alegar que recebeu uma mensagem de um processo correto, mesmo que isso nunca tenha acontecido, já com autenticação este tipo de comportamento malícioso pode ser facilmente detectado.

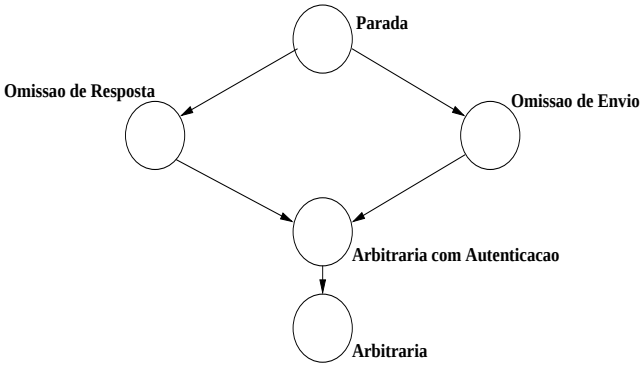

Figura 2.1: Tipos de faltas de acordo com a severidade.

Quando trabalhamos com sistemas temporizados existem dois novos tipos de faltas que podem ser consideradas:

- **Relógio:** O relógio a qual o processo tem acesso apresenta um desvio maior que o especifi cado em determinado intervalo de tempo (não satisfaz a equação 2.1);
- **Desempenho:** O processo demora mais tempo para executar um passo de processamento que seu limite máximo ou menos tempo que seu limite mínimo especifi cados (ver propriedade 3 dos sistemas síncronos).

Estes tipos de faltas existem apenas em modelos de sistemas como o síncrono puro e o assíncrono temporizado.

Vale observar que as faltas de desempenho englobam as de omissão e as de parada, que nestes sistemas temporizados passam a ser casos específi cos de faltas de desempenho onde o processo demora um tempo infi nito para executar o processamento esperado.

2.1.3 Tipos de Faltas na Comunicação

O suporte de comunicação por onde os processos do sistema enviam suas mensagens pode sofrer os seguintes tipos de falhas:

- **Parada:** Uma ligação faltosa da rede para de transportar mensagens;
- **Omissão:** A rede se omite intermitantemente de transportar mensagens através dela;
- **Arbitrária:** A rede pode exibir qualquer comportamento. Podendo, inclusive, gerar mensagens espúrias. Este tipo de falha também pode ser chamado de **maliciosa** ou **bizantina**;

Estes tipos de faltas são aplicáveis tanto a sistemas síncronos quanto a sistemas assíncronos. No caso de sistemas síncronos (ou temporizados) temos também faltas de **desempenho**, que são as faltas envolvendo comportamento temporal. Estas faltas acontecem sempre que a ligação faltosa da rede atrasa ou adianta o transporte de uma mensagem fora dos limites de tempo especifi cados.

2.1.4 Topologia da Rede

A rede de comunicação pode ser modelada como um grafo. Onde os vértices representam os processos comunicantes e as arestas representam as ligações entre estes processos. Geralmente é desejável que não existam particionamentos na rede, i.e. mesmo em caso de falha em um processo ou ligação os processos corretos continuam se comunicando.

Existem dois tipos básicos de redes: as redes ponto-a-ponto e as redes de difusão. A fi gura 2.2 apresenta estes dois tipos.

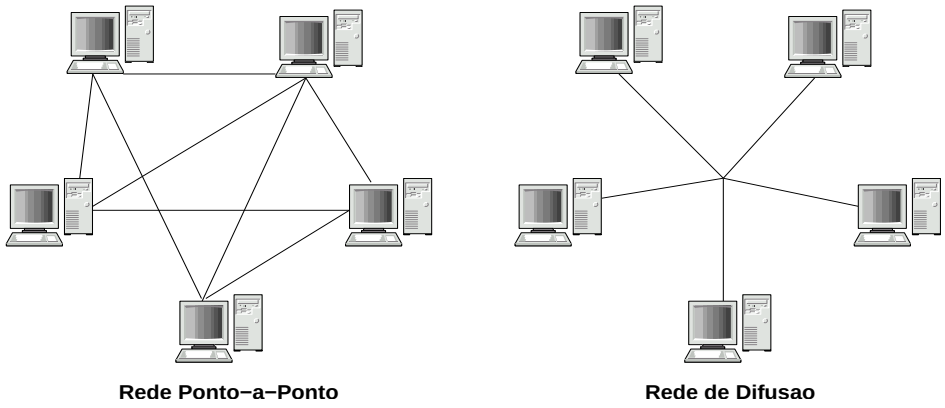

Figura 2.2: Tipos de Redes.

Na fi gura 2.2, as redes ponto-a-ponto são caracterizadas por caminhos físicos conectando processos dois a dois. Já com a topologia multiponto (rede de difusão) o meio físico compartilhado permite processos se conectando diretamente a mais de um processo. Nesta última topologia, qualquer falha de ligação só prejudica o processo conectado através desta ligação ao meio compartilhado (rede).

As redes baseadas em barramento são redes tipo difusão, já as redes em anel são redes ponto-a-ponto. A Internet, como a maioria das redes de larga escala, é uma rede de topologia mista.

2.1.5 Determinismo dos Processos

Um processo no sistema pode ser modelado como um autômato (possivelmente infi nito). Um processo determinísta tem seu estado atual defi nido apenas pelo conjunto de transições por êle executado. A execução de passos em um processo não determinísta (também chamado aleatório) resulta

em um conjunto de possíveis estados onde o processo pode se encontrar, cada um associado à uma probabilidade.

2.2 Suporte de Comunicação de Grupo (SCG)

Um grupo é uma coleção de objetos (ou processos), e um suporte de comunicação de grupo é a funcionalidade no sistema distribuído reponsável pelo gerenciamento e comunicação entre membros do grupo.

Para que o suporte de comunicação de grupo (SCG) possa garantir as propriedades exigidas pelas primitivas de comunicação de grupo oferecidas, ele deve fornecer uma visão consistente do grupo.

Esta seção apresenta apresenta alguns conceitos importantes tanto na comunicação como na gestão de grupos.

2.2.1 *Membership*

Um problema fundamental relacionado a sistemas distribuídos, em especial em suportes de comunicação de grupo, diz respeito à determinação, em tempo de execução, de quais processos são membros de um grupo. Esta lista de membros pertencentes ao grupo é o que chamamos comumente de *membership* (pertinência) ou visão.

O *membership* de um grupo pode ser estático ou dinâmico. Em um *membership* estático os membros do grupo são conhecidos previamente, em tempo de compilação. Em *memberships* dinâmicos, processos entram e saem de grupos, o quê, representa uma pertinência que evolui, possuindo um número de membros variável em função do tempo.

Os SCG trabalham normalmente com *membership* dinâmico por questões de flexibilidade. O conjunto de membros de um grupo pode mudar por pelo menos três razões:

- **Requisitos da Aplicação:** Algum requisito funcional da aplicação atuando sobre o SCG deve implicar nas alterações de *membership*. Por exemplo, em uma vídeo-conferência, algumas pessoas podem entrar na sessão (grupo) após o início desta, e ainda algumas podem deixá-la antes de seu fim;
- **Falhas de Processos:** Processo falhos devem ser removidos de grupos;
- **Redimensionamento do Sistema:** Grupos podem crescer ou diminuir segundo demandas. As vezes a demanda pelo sistema cresce ou diminui tanto que um redimensionamento de seus recursos se faz necessário.

Nem todo SCG tem acesso e/ou fornece o *membership* dos grupos por ele gerenciado, alguns suportes funcionam sem este tipo de informação. Quando o *membership* é disponibilizado pelo SCG, esta lista de membros do grupo é tornada disponível através do Serviço de *Membership* de Grupo (SMG).

Em *membership* dinâmico as alterações no grupo são controladas pelo SMG, que processa as mudanças e altera a lista de membros do grupo, através de operações oferecidas aos processos, como por exemplo *join* e *leave*.

Num ambiente livre de falhas de processos as operações citadas seriam suficientes para que o SMG tivesse uma visão correta do grupo, entretanto nem sempre esse é o caso, e na maior parte dos sistemas reais, as falhas não podem ser desconsideradas. Assim, cabe ao SMG, através de algum mecanismo de detecção de falhas, descobrir os processos faltosos de modo a excluí-los da lista de pertinência, mantendo assim um *membership* consistente.

2.2.2 Detectores de Falhas

Um **detector de falhas** é, de maneira simplificada, o módulo do SCG responsável pela monitoração do comportamento dos membros do grupo. A detecção de falhas é baseada normalmente na associação de cada membro de um grupo a um detector específico (seu detector). Cada um destes detectores de falhas monitora os membros, ou um subconjunto dos membros, de um grupo e mantém uma lista de processos suspeitos (membros sob suspeita).

Em [9] é estudada a semântica destes detectores. A taxonomia introduzida neste texto leva a definição de classes de detectores, tomando como base as seguintes propriedades:

- **Completude (*Completeness*):** Define que os processos faltosos serão suspeitos por todos processos corretos do sistema;
- **Corretude (*Acuracy*):** Define que os processos corretos não serão suspeitos por nenhum processo correto do sistema.

O enfraquecimento destas propriedades dá origem a 8 detectores de falhas. Sendo de especial interesse os detetectores $\mathcal{P}$ (Perfeito) e os de semântica $\diamond\mathcal{W}$ (Eventualmente Fraco). Um detector $\mathcal{P}$ não comete erros de nenhum tipo e só pode ser implementado em sistemas completamente síncronos. O detector $\diamond\mathcal{W}$ pode cometer erros tanto em termos de completude quanto em termos de corretude, assim ele pode suspeitar de processos corretos e não suspeitar de processos falhos.

O detector $\diamond\mathcal{W}$ é o detector de faltas mais fraco que pode ser utilizado para implementar consenso e difusão atômica (ver próxima seção) [7], pelo menos teoricamente [15].

Modelos de sistemas distribuídos que fazem uso de detectores de falhas são alternativas a modelos síncronos e assíncronos e, ainda, equivalentes a modelos intermediários como o de sincronismo parcial [9]. Quanto mais completo e correto o detector do sistema for, mais este se aproxima de um sistema síncrono. Por outro lado, quanto mais brandas forem as semânticas de completude e corretude de um detector mais o sistema se aproxima dos modelos assíncronos.

2.3 Primitivas de Comunicação de Grupo

A comunicação de grupo tem se mostrado um paradigma necessário em sistemas distribuídos. Suportes de comunicação de grupo disponibilizam várias primitivas de comunicação com diferentes garantias de confiabilidade e ordenação, atendendo diferentes requisitos de aplicações. Estas garantias são definidas através das propriedades implementadas pelos serviços que oferecem estas primitivas no suporte.

Em um sistema distribuído onde os processos se comunicam por difusão (seletiva ou não), a presença de faltas (tanto nas comunicações quanto nos processos) pode ocasionar perdas de mensagens, que levam a inconsistências nos estados dos membros dos grupos. Assim sendo, as primitivas utilizadas para comunicação devem oferecer pelo menos confiabilidade (que será definida mais adiante) e um nível de garantia em relação à ordenação das mensagens.

Existem quatro tipos básicos de comunicação de grupo com garantias em sistemas assíncronos. Nesta seção elas serão apresentadas conforme definido em [23], onde uma ênfase especial será dada à difusão confiável, que é o objeto de estudo neste trabalho.

2.3.1 Difusão Confiável

O primeiro serviço de difusão a ser apresentado é o de difusão seletiva confiável[2] (*Reliable Multicast*). De maneira simples, um serviço de difusão confiável garante que todas as mensagens enviadas a um grupo de processos serão recebidas por todos os membros não faltosos do grupo.

Se considerarmos $\mathcal{M}$ como o conjunto de todas as possíveis mensagens do sistema e $\mathcal{P}$ como o conjunto de todos os possíveis processos do sistema, podemos introduzir a difusão confiável em termos de duas primitivas: *R-multicast*(G,m)e *R-deliver*(m)onde m é uma mensagem ($m \in \mathcal{M}$) e G é um conjunto de processos ($G \subseteq \mathcal{P}$), também chamado de grupo. As primitivas são definidas da seguinte forma:

- *R-multicast*(G,m): A mensagem m é difundida para todos os processos pertencentes ao grupo G.

[2] que chamaremos neste texto simplesmente de difusão confiável.

- *R-deliver*(m): A mensagem *m* é liberada para a aplicação.

Estas primitivas devem satisfazer as seguintes propriedades:

1. **Validade:** Se um processo correto difundiu *m* em *G*, então algum processo correto pertencente à *G* entregará *m* ou nenhum processo do grupo está correto;
2. **Acordo:** Se um processo correto pertencente a *G* entrega a mensagem *m*, então todos os processo corretos pertencentes a *G* entregarão *m*;
3. **Integridade:** Para qualquer mensagem *m*, cada processo correto pertencente a *G* entrega *m* no máximo uma vez e apenas se *m* foi previamente difundida em *G*.

A propriedade de integridade está associada a condição de *safety* (correção) enquanto as duas outras representam a condição de *liveness* (terminação).

Os algoritmos de difusão confiável presentes na literatura geralmente partem da premissa de que a comunicação multiponto será construída utilizando-se de serviços de comunicação ponto-a-ponto confiáveis. Por exemplo, o algoritmo apresentado em [9, 23] parte de duas primitivas de comunicação básicas:

- *send*(m, p): Envia a mensagem *m* para o processo p ($p \in \mathcal{P}$).
- *receive*(m): Recebe a mensagem *m*.

E assume que estas primitivas de comunicação satisfazem as seguintes propriedades:

1. **Validade (*Liveness*):** Se p envia *m* para q ($q \in \mathcal{P}$), e se p, q e a ligação entre eles estiverem corretos, então q recebe *m*;
2. **Integridade Uniforme (*Safety*):** Para qualquer mensagem *m*, q recebe *m* pelo menos uma vez de p e apenas se p enviou *m* a q.

O referido algoritmo de difusão confiável é apresentado a seguir (algoritmo 1). Este algoritmo garante difusão seletiva confiável, tomando como base o serviço de comunicação ponto-a-ponto confiável. Na notação utilizada temos a mensagem $m \in \mathcal{M}$ e os processos $p, q \in G$.

A idéia básica neste algoritmo é que ao receber uma mensagem pela primeira vez, cada processo a retransmite para todos os membros do grupo menos para si mesmo. Partindo-se da premissa de que existem canais ponto-a-ponto com as propriedades enunciadas acima conectando todos os membros do grupo e que a ocorrência de faltas em nós ou ligações nunca gera um grafo desconexo com os nós membros e as ligações restantes do grupo, é fácil demonstrar que nenhum membro correto do grupo ficará sem pelo menos uma copia de cada mensagem difundida.

Algoritmo 1 Algoritmo de difusão confiável

```
1: {To execute R-multicast(G, m)}
2: for all p ∈ G do
3:     send(m, p)
4: end for
5:
6: {To realize R-deliver(m)}
7: receive(m) for the first time
8: if sender(m) ≠ p then
9:     R-multicast(G, m)
10: end if
11: R-deliver(m)
```

2.3.2 Difusão FIFO

A difusão seletiva FIFO (*First-In-First-Out*) nada mais é que uma difusão confiável com propriedades de ordenação FIFO. Existem duas propriedades referentes à ordenação FIFO, conforme especificado a seguir [23]:

- **Ordenação FIFO Global:** Se um processo difunde uma mensagem m antes de difundir m', então todos os processos corretos em G não entregam m' antes de entregar m.
- **Ordenação FIFO Local:** Se um processo difunde uma mensagem m em G antes de difundir m' em G, então todos os processos corretos em G não entregam m' antes de entregar m.

A ordenação FIFO garante que as mensagens difundidas por um processo serão entregues pelos receptores na mesma ordem em que foram realizadas. A diferença entre as duas propriedades de ordenação FIFO apresentadas está no fato de uma delas garantir ordenação apenas nas mensagens endereçadas ao grupo (FIFO local) enquanto a outra garante ordenação FIFO para todas as mensagens independentemente do grupo em que elas foram difundidas.

2.3.3 Difusão Causal

A ordenação FIFO é adequada apenas quando o contexto de uma mensagem m consiste apenas de mensagens difundidas pelo mesmo emissor. Entretanto, se o contexto de m depende também das mensagens entregues pelo seu emissor então a ordenação FIFO não é mais suficiente. Nestes casos, faz-se necessário um tipo de ordenação que leve em consideração a precedência causal de eventos[3] [27]. Dizemos que o evento e precede causalmente o evento f (denotado $e \rightarrow f$) se e somente se:

[3]Os eventos considerados em sistemas distribuídos são os passos de processamento e as ativações de primitivas de comunicação.

1. o mesmo processo executa e e depois executa f, ou;
2. e é a difusão de uma mensagem e f é a entrega desta mensagem, ou;
3. existe um evento h, tal que $e \rightarrow h$ e $h \rightarrow f$.

Por esta definição pode-se perceber que a relação de precedência causal é acíclica e transitiva.

A ordenação causal reforça a ordenação FIFO generalizando a noção de dependência entre mensagens e garantindo que uma mensagem só será entregue a aplicação se a mensagem que a causou tiver sido entregue antes. Formalmente, uma difusão causal é caracterizada por uma difusão confiável que satisfaz uma das seguintes propriedades de ordenação:

- **Ordenação Causal Global:** Se a difusão de uma mensagem m precede causalmente a difusão de uma mensagem m', então nenhum processo correto entrega m' antes de entregar m.
- **Ordenação Causal Local:** Se a difusão de uma mensagem m em G precede causalmente a difusão de uma mensagem m' em G, então nenhum processo correto em G entrega m' antes de entregar m.

A diferença entre a ordenação causal global e a local está, assim como na ordenação FIFO, no fato da primeira garantir a precedência causal entre mensagens além dos limites do grupo, enquanto a segunda só garante esta precedência para mensagens difundidas em determinado grupo.

2.3.4 Difusão Atômica

Se a difusão de duas mensagens não esta causalmente relacionada, a difusão causal não impõem nenhum tipo de restrição quanto a ordem de entrega das mensagens e permite inclusive que elas sejam entregues em ordens diferentes em diferentes processos. Para algumas aplicações este comportamento é inaceitável, principalmente devido à possíveis inconsistências que ele pode levar.

Para evitar esse tipo de problema, a difusão atômica garante que todos os processos corretos entregarão todas as mensagens na mesma ordem. Desta forma, todos os processos tem a mesma visão [3] do sistema e podem agir de maneira consistente sem comunicações adicionais.

Formalmente, uma difusão atômica é uma difusão confiável que satisfaz uma das seguintes propriedades de ordenação:

- **Ordenação Total Local:** Se dois processos corretos p e q entregam as mensagens m e m' endereçadas ao grupo G, então p entrega m antes de m' se e somente se q entregar m antes de m'.

- **Ordenação Total Global:** Se dois processos corretos p e q entregam as mensagens m e m', então p entrega m antes de m' (denotado $m < m'$) se e somente se q entregar m antes de m'. Além disso $<$ é uma relação acíclica.

Se uma difusão satisfaz a ordenação total local ela é chamada difusão total local, e se ela satisfaz a ordenação total global ela é chamada, consequentemente, de difusão total global.

Em [23] é definido uma outra propriedade de ordenação atômica mais forte que a local e mais fraca que a global: a *Pairwise Atomic Order*. Esta propriedade é a própria ordenação total global sem a restrição de aciclicidade no operador $<$ (permitindo, por exemplo, inconsistências em grupos sobrepostos).

2.3.5 Reforçando a Difusão Atômica

Apesar de definir que os processos de um grupo (local) ou de todo sistema (global) devem entregar as mensagens difundidas em uma mesma ordem, a difusão atômica não define que ordem é essa. Assim pode-se combinar esta difusão com outros tipos de ordenação a fim de garantir a atomicidade da entrega das mensagens em uma determinada ordem. Os duas principais combinações que podem ser feitas são[4]:

- **Difusão Atômica FIFO:** Difusão FIFO + Ordenação Total.
- **Difusão Atômica Causal:** Difusão Causal + Ordenação Total.

Os relacionamentos entre todas as primitivas de difusão apresentadas nesta seção esta resumido na figura 2.3 que desconsidera as variações locais e globais definidas para os protocolos de difusão. Se considerarmos estas variações temos um total de 13 primitivas ao invés de 6. A difusão atômica causal é considerada o tipo de difusão mais "forte"se desconsiderarmos requisitos temporais.

2.3.6 Difusão Temporizada

Muitas aplicações requerem a entrega de uma mensagem por todos os membros de um grupo G com um atraso máximo de Δ_G unidades de tempo. Esta propriedade é chamada de **Temporização-Δ_G**. Este tipo de difusão só faz sentido em sistemas temporizados, onde pode-se estabelecer "limites"desejáveis para a utilização de serviços. O parâmetro Δ_G é chamado a latência da difusão temporizada no grupo G.

[4]Desconsiderando as variações locais e globais de cada propriedade.

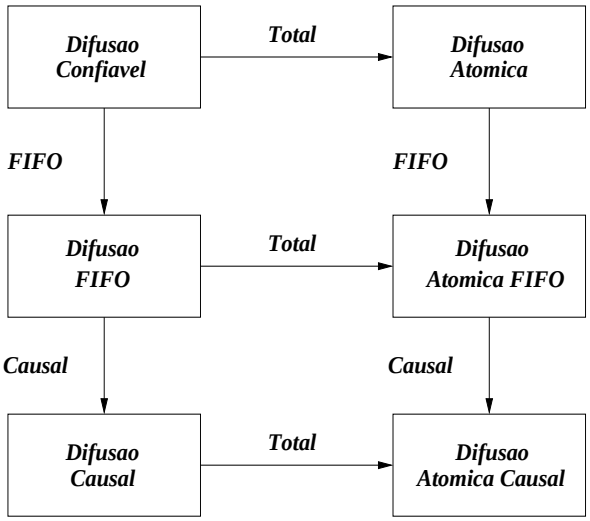

Figura 2.3: Relacionamento entre as primitivas de *multicast*.

Como em sistemas distribuídos existem duas interpretações para o tempo: tempo real, medido por um observador externo, e tempo local, medido pelos relógios locais dos processos, temos duas propriedades diferentes de temporização-Δ_G:

- **Temporização-Δ_G com Tempo Real:** Existe uma constante conhecida Δ_G de tal forma que se uma mensagem m é difundida em G em um instante de tempo real t, nenhum processo correto pertencente a G entregará m após o instante $t + \Delta_G$ de tempo real.
- **Temporização-Δ_G com Tempo Local:** Existe uma constante conhecida Δ_G de tal forma que nenhum processo correto p ($p \in G$) entrega m (difundida em G) depois do instante de tempo $ts(m) + \Delta_G$ medido no relógio de p.

Na definição da temporização-Δ_G com tempo local, é assumido que cada mensagem m leva consigo o seu *timestamp* (instante em que foi difundida medido no relógio local do emissor), denotado por $ts(m)$.

Da mesma forma que a difusão atômica, a difusão temporizada também pode ser integrada a todos os outros tipos de difusões, inclusiva à difusão confiável, que neste caso seria chamada de *Difusão Confiável Temporizada*.

2.3.7 Difusão Uniforme

Todas as propriedades requeridas pelos tipos de difusão apresentadas até agora nada dizem a respeito do comportamento de processos faltosos que eventualmente fazem parte dos grupos, em

especial, estamos interessados em quando um processo faltoso faz a entrega de uma mensagem. Desta forma é possível fortalecer as propriedades apresentadas até aqui de tal forma a considerar a presença de processos faltosos no grupo.

Como exemplo, apresentamos o fortalecimento das propriedades de acordo e integridade das difusões para o tratamento de processos faltosos:

1. **Acordo Uniforme:** Se um processo (correto ou faltoso) pertencente a *G* entrega a mensagem *m*, então todos os processo corretos pertencentes a *G* entregarão *m*;
2. **Integridade Uniforme:** Para qualquer mensagem *m*, cada processo (correto ou faltoso) pertencente a *G* entrega *m* no máximo uma vez e apenas se *m* foi previamente difundida em *G*.

O uso de propriedades uniformes evita que processos faltosos originem inconsistências em grupos de processos.

Além dessas propriedades, é possível também fortalecer com a uniformidade as propriedades de ordenação e temporização. Para um tratamento completo ver [23].

2.4 Conclusão

Este capítulo apresentou os príncipais conceitos envolvidos na especificação de sistemas de comunicação de grupo e sistemas distribuídos de um modo geral.

Foram apresentados os principais parâmetros que definem o ambiente computacional onde o SCG será utilizado. Este ambiente é de fundamental importância para a definição dos "limites"dos serviços oferecidos pelo SCG.

Os serviços de *membership* e detecção de falhas trabalham de maneira integrada com SCG a fim de manter uma visão consistente do grupo, porém foi visto que nem todos os SCG mantém ou disponibilizam estes serviços.

As primitivas de comunicação apresentadas neste capítulo definem as diferentes qualidades de serviço que podem ser oferecidas pelos sistemas comunicação de grupo. Dentre estas, destaca-se a difusão confiável como o mecanismo mais básico de comunicação, sendo usado por todos os outros serviços apresentados para garantir algum tipo de propriedade.

No próximo capítulo serão apresentados diversos protocolos que visam prover difusão confiável em redes de larga escala, utilizando a tecnologia *multicast* IP para a obtenção de um melhor desempenho.

Capítulo 3

Comunicação de Grupo em Larga Escala

Com o advento da Internet surgiu a necessidade de se adaptar alguns conceitos de sistemas distribuídos considerando a escala de uma rede mundial. As soluções e algoritmos próprios para sistemas distribuídos precisam atender requisitos de escalabilidade envolvendo tanto a distância física entre os componentes do sistema quanto o número de componentes participantes.

Desta forma vários modelos e algoritmos para comunicação de grupo tiveram de ser reformulados para funcionarem adequadamente em ambientes de larga escala, principalmente no sentido de explorar as facilidades disponíveis em níveis de enlace e rede nos protocolos largamente utilzados na Internet.

Este capítulo apresenta o *multicast* IP como o mecanismo básico de comunicação de grupos na Internet e também os princípais tipos de protocolos de difusão confiável encontrados na literatura baseados em *multicast* IP.

3.1 *Multicast* IP

Usualmente as transmissões na Internet são feitas entre um emissor e um receptor, entretanto, para muitas aplicações, faz-se necessário um mecanismo de comunicação que permita que mensagens de um emissor sejam recebidas por diversos receptores (comunicação de um para muitos). Dentre os exemplos destas aplicações podemos citar aplicações de bancos de dados distribuídos, teleconferência, servidores replicados, etc.

A fim de atender os requisitos destas aplicações, o IETF (*Internet Engineering Task Force*) propôs através dos RFCs 966 [14] e 988 [13] uma extensão ao protocolo IP que permite a difusão de pacotes para grupos de *hosts*. Este conjunto de extensões deram origem ao *multicast* IP, um mecanismo através do qual é possível a um *host* emissor enviar um datagrama a um conjunto de zero ou mais

hosts receptores, denominado grupo. Cada grupo é identificado por um endereço IP (de maneira semelhante a um *host*). Quando o destino de um datagrama é um grupo, é feita uma tentativa de entregá-lo a todos os membros deste, entretanto, a exemplo do *unicast* IP não existem quaisquer garantias para:

- Entrega dos pacotes a todos os membros do grupo;
- Integridade dos pacotes;
- Entrega dos pacotes na ordem de envio pelo emissor;
- Entrega dos pacotes em ordem única para todos os receptores.

As extensões propostas no *multicast* IP acrescentam apenas o endereçamento de grupos ao protocolo de rede IP, mantendo o restante das semânticas asssociadas a este protocolo intactas.

Nas próximas seções serão apresentados os principais elementos do *multicast* IP, destacando-se o modelo de grupo utilizado, as extensões ao protocolo IP e o protocolo IGMP (*Internet Group Management Protocol*), utilizado para gerenciamento de grupos IP.

3.1.1 Modelo de Grupos no *Multicast* IP

O endereçamento de grupos, conforme já apresentado, é realizado através de uma classe específica de endereços IP, a classe D. Existem dois tipos de grupos: grupos permanentes e grupos temporários. Os grupos permanentes estão sempre presentes e não precisam ser criados, já os temporários devem ser criados e duram enquanto houverem processos pertencentes a eles (a duração do programa distribuído). Exemplos de grupos permanentes são o grupo de *hosts* de uma rede local (endereço `224.0.0.1`) e o grupo de todos os roteadores de uma rede local (endereço `224.0.0.2`).

Os grupos definidos no *multicast* IP são grupos abertos no sentido de que um emissor não pertencente ao grupo pode enviar mensagens ao mesmo.

O gerenciamento dos membros de um grupo é feito de maneira dinâmica, de tal forma que a associação de endereços IP à *hosts* seja bastante flexível. Assim, um *host* pode pertencer a um ou mais grupos, ou a nenhum grupo em determinado momento. Esta associação dinâmica é realizada através de três operações que o módulo IP deve prover:

1. *CreateGroup*: Cria um grupo tempórario (transiente) de *hosts*, cujo o *host* invocador é o único membro;
2. *JoinGroup*: Pede a adição do *host* invocador ao grupo especificado (seja ele permanente ou temporário);

3. *LeaveGroup*: Pede a remoção do *host* invocador do grupos especifi cado.

Note que não existe operação para destruição de grupos. O que se justifi ca pelo fato de grupos permanentes não poderem ser destruídos e grupos temporários só existirem enquanto seu número de membros é maior que zero.

As operações de envio (*send*) e recepção (*receive*) de pacotes são as mesmas do *unicast* IP. A única diferença é que o parâmetro que especifi ca endereços deve suportar endereços de grupos.

3.1.2 Extensões ao Protocolo IP

Esta seção discute as modifi cações e extensões necessárias ao protocolo IP para o suporte ao modelo de grupo de *hosts*.

3.1.2.1 Endereçamento de Grupos

Conforme já apresentado, cada grupo têm um endereço IP de classe D (compreendido pelo intervalo de `224.0.0.0` a `239.255.255.255`) associado a ele (ver fi gura 3.1), desta forma, para um emissor enviar um datagrama a um grupo, basta que ele defi na o endereço IP do grupo no campo destino do datagrama IP.

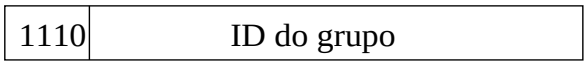

Figura 3.1: Anatomia de um Endereço IP Classe D.

Outro serviço importante é a tradução de endereços *multicast* IP para endereços *multicast* de enlace. Este serviço, que deve ser provido pela camada de enlace, deve permitir, por exemplo, o mapeamento de endereços *multicast* IP para endereços *multicast ethernet*. Para redes que não suportam este mapeamento direto, os endereços *multicast* IP devem ser mapeados para endereços de *broadcast*, de tal forma que todos os *hosts* recebam o pacote e os que não pertencem ao grupo os descarte.

3.1.2.2 Gerenciamento de Grupos

Para suportar o gerenciamento dinâmico de grupos através das operações *CreateGroup*, *JoinGroup* e *LeaveGroup*, o módulo IP de cada *host* deve interagir com um ou mais agentes de *multicast*. Estes agentes, realizam a tarefa de criar e manter grupos transientes no contexto da Internet. Assim, é responsabilidade destes agentes saber quais grupos possuem em seus domínios de rede membros, de forma a receber pacotes endereçados aos mesmos. É papel destes agentes, também, conhecer os

demais domínios de rede (e os agentes correspondentes) que contém membros de um dado grupo. Desta forma é fácil perceber que deve existir pelo menos um agente deste tipo em cada rede IP que suporte *multicast*, e usualmente eles são implementados em roteadores ou *gateways* por questões de desempenho, já que eles fi cam mais perto da entrada/saída da sub-rede correspondente.

As interaçõs entre os módulos IP e o agente *multicast* de sua rede se dão através do Protocolo de Gerenciamento de Grupos da Internet (IGMP- *Internet Group Management Protocol*). Este protocolo é apresentado na próxima sub-seção.

3.1.2.3 Controle de Distância

Conforme já citado, os membros de um grupo de *hosts* podem estar espalhados por diversos domínios na Internet (sub-redes). Assim sendo, a entrega de pacotes a estes membros utilizará o núcleo da rede através do roteamento dos pacotes pelos agentes *multicast*. Para evitar que um pacote vague indefi nidamente pelo conjunto de roteadores do núcleo da rede, o campo `Time-To-Live` do pacote IP é normalmente usado para defi nir um raio máximo para a entrega do pacote.

Uma outra forma de controle de distância que o *multicast* IP defi ne é a idéia de **distância administrativa**. Nesta medida, o escopo do *multicast* é defi nido em termos administrativos, como por exemplo "este departamento", "este prédio", "esta empresa", e assim por diante. Este tipo de medida de distância é implementado também através do campo `Time-to-Live`, utilizando uma tabela de valores convenientemente preparada.

3.1.3 IGMP - *Internet Group Management Protocol*

O IGMP é utilizado para comunicação entre *hosts* e seus agentes *multicast* no suporte à criação e manutenção de grupos.

O IGMP é um protocolo assimétrico [13], e é descrito aqui sob o ponto de vista do *host*. Assim como o ICMP (*Internet Control Message Protocol*), o IGMP é parte integrante do módulo IP, e deve ser implementado por todos os *host* que participam completamente do modelo *multicast* IP. As mensagens IGMP tem um formato bem defi nido e devem ser encapsuladas em pacotes IP tradicionais. O formato das mensagens IGMP é defi nido na fi gura 3.2.

Os campos apresentados na fi gura 3.2 são detalhados a seguir:

- *Type*: Tipo da mensagem. Os oito tipos de mensagens IGMP defi nidos são apresentados na tabela 3.1.

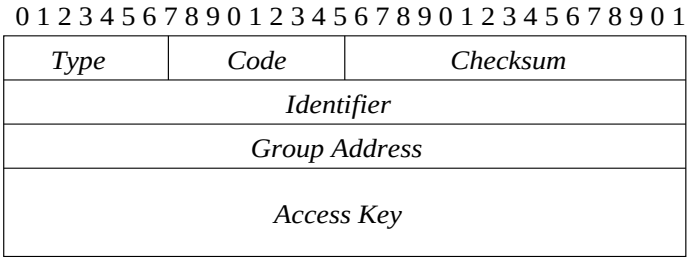

Figura 3.2: Formato das Mensagens IGMP.

Type	**Mensagem**
1	requisição para a criação de um grupo
2	resposta para a criação de um grupo
3	requisição para a adição de um *host* ao grupo
4	resposta para a adição de um *host* ao grupo
5	requisição para a remoção de um *host* ao grupo
6	resposta para a remoção de um *host* ao grupo
7	confi rmação de uma requisição
8	confi rmação de uma resposta

Tabela 3.1: Tipos de Mensagens IGMP.

- *Code*: Na mensagem de criação de grupo este campo defi ne se o grupo é público (qualquer *host* pode entrar no grupo) ou restrito (necessita-se de uma chave para entrar no grupo). Este campo também é utilizado para especifi car o conteúdo das respostas.

- *Checksum*: Este campo de 16 bits defi ne a soma de verifi cação e assegura a integridade da mensagem.

- *Identifier*: Este campo é o identifi cador da mensagem. Este identifi cador serve para distinguir uma requisição de outras vindas do mesmo *host*. Em respostas, este campo contém a mesmo valor da mensagem que causou. A única excessão a esta regra ocorre na resposta a uma requisição de criação de grupo, quando o valor deste campo deve ser zero.

- *Group Address*: O endereço do grupo. Em todas as respostas este campo deve ser igual ao especifi cado na requisição, excetuando-se na resposta a requisição da criação de grupo, que se bem sucedida trás o endereço do novo grupo criado, caso contrário retorna zero.

- *Access Key*: Este campo defi ne a chave de acesso de 64 bits para grupos privados. Nas mensagens de criação de grupo e nas mensagens que dizem respeito a grupos públicos este campo deve ser zero. Nas mensagens de resposta a criação de grupos transientes e nas mensagens que dizem respeito a grupos privados este campo deve ser defi nido com uma chave de acesso diferente de zero.

O funcionamento do protocolo é bastante simples: mensagens de requisição são enviadas somente por *hosts* e mensagens de resposta são enviadas apenas por agentes *multicast*. Para todas as mensagens definidas o campo `Time-to-Live` do pacote IP que contém a mensagem deve ser definido como 1, especificando que a mensagem deve ter apenas escopo local (em nível de rede local). Para uma completa descrição do IGMP ver [13].

3.1.4 Aspectos de Implementação

Uma pilha TCP/IP com suporte a *multicast* é apresentado na figura 3.3.

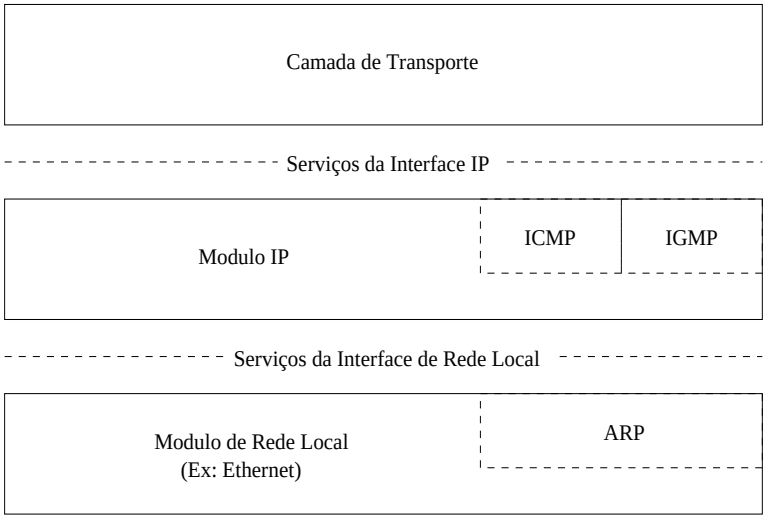

Figura 3.3: Arquitetura padrão de um módulo IP.

No modelo da figura 3.3, tanto o ICMP quanto o IGMP são implementados na camada IP, já o mapeamento de endereços IP para endereços de rede local é responsabilidade da camada de enlace, através do ARP (*Address Resolution Protocol*).

O módulo IP provê vários serviços para a camada de transporte, entre eles destacam-se o `send` e o `receive` usados para envio e recepção de dados respectivamente, e os serviços de *multicast*: `CreateGroup`, `JoinGroup` e `LeaveGroup`.

Existem três níveis de conformidade com o modelo de *multicast* IP, e cada nível destes requer a implementação de alguns dos módulos presentes na figura 3.3. A tabela 3.2 mostra estes níveis.

Uma implementação de nível 0 não requer a implementação de nenhum módulo adicional. Os datagramas recebidos que sejam endereçados a grupos devem ser descartados pelo módulo IP. No nível 1 nenhuma alteração deve ser realizada na interface entre a camada IP e a camada de transporte, a única implementação adicional necessária é a tradução de endereços *multicast* que deve ser realizada pelo ARP na camada de rede local. Uma implementação que atenda ao nível 2 do modelo, além de

Nível	Descrição
0	Sem suporte a multicast IP
1	Envia mas não recebe datagramas *multicast* IP
2	Envia e recebe datagramas *multicast* IP

Tabela 3.2: Níveis de conformidade *Multicast* IP.

prover todos os serviços dos níveis anteriores, deve implementar o protocolo IGMP, que fornece os serviços de suporte à grupos para a camada de transporte.

3.1.5 MBone

Apesar do *multicast* IP ser uma tecnologia relativamente madura, a mesma ainda não se faz presente em todas as redes na Internet. Isto se deve principalmente à idéia, muitas vezes errônea, de que a utilização desta tecnologia causa um grande tráfego na rede. Entretanto, existe um *backbone* já consolidado que otimiza o tráfego *multicast* na rede; este conjunto de roteadores que entende pacotes IP endereçados a grupos é chamado de MBone (*virtual Multicast Backbone On the interNEt*) [26].

O MBone foi criado em 1992 a partir de um pequeno conjunto de roteadores no núcleo da Internet, desde então o mesmo tem crescido espantosamente, espalhando-se por todos os continentes. Os roteadores participantes do MBone (chamados *mrouters*) utilizam algoritmos de roteamento como o DVMRP (*Distance Vector Multicast Routing Protocol*), o MOSPF (*Multicast Open Shortest Path First*) ou o PIM (*Protocol Independent Multicast*) para distribuir os pacotes *multicast* IP aos grupos espalhados pela rede. A comunicação entre os roteadores é feita através de túneis que ligam "ilhas"*multicast*, conforme apresentado na figura 3.4.

As "ilhas"*multicast* da figura 3.4 são sub-redes capazes de realizar *multicast* internamente. A figura 3.4 ilustra túneis formados para a concretização da difusão de mensagens da rede 1 até as redes 2 e 3. Todos os roteadores que estão nas pontas destes túneis são *mrouters*.

O MBone tem sido utilizado geralmente na realização de videoconferências e transmissão de eventos pela Internet. Entretanto, suas capacidades podem também ser utilizadas, de forma generalizada, nas comunicação de sistemas cujos componentes estejam distribuídos pela Internet.

3.2 Algoritmos de Difusão Confiável

Quando se usam redes de larga escala, em especial a Internet, o ponto chave é diminuir o número de interações entre os componentes do sistema distribuído. A diminuição de interações entre componentes melhora o desempenho do sistema: são menores as dependências de carga da rede, menores

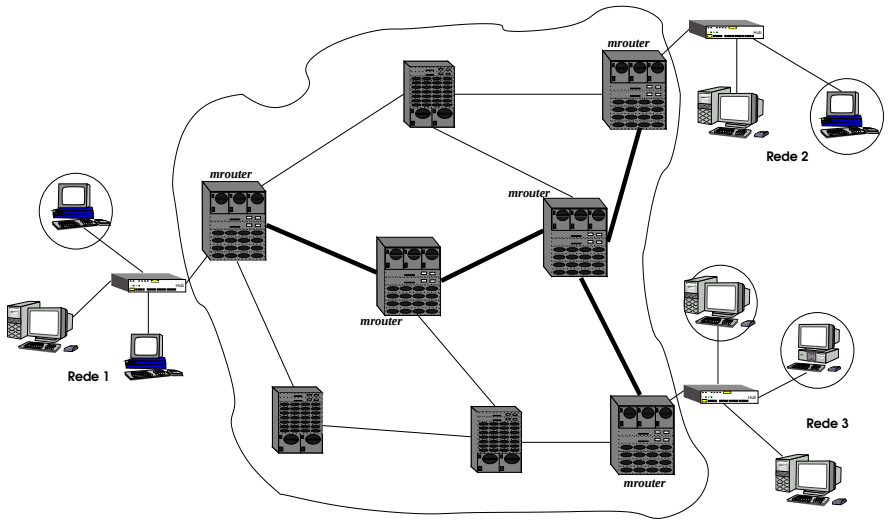

Figura 3.4: Túnel MBone ligando "ilhas"*multicast*.

as perdas de mensagens, etc. Esta estratégia é aplicada, por exemplo, em suportes de grupo onde as comunicações ponto-a-ponto são substituídas por difusões seletivas, como o *multicast* IP no caso da Internet, diminuindo drásticamente o número de pacotes trafegando na rede.

Esta seção apresenta uma abstração para o serviço de difusão não confiável do *multicast* IP e em seguida as várias classes de protocolos de difusão confiável escaláveis construídos sobre este.

3.2.1 Difusão Não Confiável

Na sequência mostraremos esforços da literatura e de nosso trabalho no sentido do uso do *multicast* IP como base de protocolos de difusão na comunicação de grupo. Começamos com uma tentativa de adaptar o protocolo de difusão confiável apresentado no algoritmo 1 sobre o *multicast* IP. Antes disto, entretanto, formalizaremos as primitivas do *multicast* IP como uma difusão não confiável:

- *U-multicast*$(id(G), m)$: é a primitiva que difunde m em G com a possibilidade de que todos os processo pertencentes a G recebam m;
- *U-receive*$(id(G), m)$: é a primitiva de recepção de m por p ($p \in G$), quando m é difundida em G.

Vale ressaltar que $id(G)$ denota um identificador único para o grupo G, como por exemplo, o endereço *multicast* IP do grupo. O serviços assim introduzido satisfaz as seguintes propriedade:

1. **Integridade:** Para qualquer mensagem m, um processo $p \in G$ recebe m endereçada a G se m foi previamente difundida em G;

2. **Sem Duplicação:** Cada mensagem m difundida em um grupo G é recebida no máximo uma vez por cada processo $p \in G$.

O algoritmo 1 tem suas premissas baseadas em comunicações ponto-a-ponto, se adaptarmos a idéia deste algoritmo usando as propriedades fracas do protocolo não confiável introduzido acima temos o algoritmo adaptado a seguir:

Algoritmo 2 Algoritmo de difusão confiável adaptado (incorreto).

1: {To execute *R-multicast*(G,m)}
2: *U-multicast*$(id(G),m)$
3:
4: {To realize *R-deliver*(m)}
5: *U-receive*$(id(G),m)$ for the first time
6: **if** *sender*$(m) \neq p$ **then**
7: *R-multicast*(G,m)
8: **end if**
9: *R-deliver*(m)

Obviamente o algoritmo 2 não implementa uma difusão confiável pois não atende as propriedades necessárias para este serviço.

Prova:

Esta prova é concretizada por contra-exemplo. Considere um serviço de difusão confiável onde um processo p execute, em um grupo G a primitiva *R-multicast*(G,m), segundo o algoritmo 2. Neste algoritmo a difusão é baseada na primitiva *U-multicast*$(id(G),m)$e como esta primitiva não oferece garantia de entrega de mensagens, m pode não ser entregue a algum processo de G, o que fere a propriedade da validade e portanto descaracteriza o mesmo como uma difusão confiável.

Apesar desta solução simples (algoritmo 2) não fornecer garantia de entrega, o reenvio sistemático de mensagens a um grupo pode melhorar em muito a confiabilidade de um sistema com características assíncronas, especialmente se estas mensagens forem enviadas repetidas vezes em intervalos de tempo variados. Entretanto, soluções melhores podem ser encontradas, principalmente em termos de desempenho do algoritmo.

Considerando ou não a natureza das primitivas de difusão não confiável introduzidas acima, e que serão usadas neste trabalho, demonstrar as propriedades da difusão confiável é praticamente impossível para sistemas completamente assíncronos. Não obstante, as próximas sessões apresentam modelos de protocolos mais confiáveis baseados nestas primitivas não confiáveis. Estes protocolos, apresentam as propriedades de difusão confiável em determinadas condições de sincronismo parcial - condições estas comumente encontradas em sistemas reais.

3.2.2 Protocolos de Difusão Confiável Escaláveis

Se com as premissas fracas apresentadas na seção anterior não podemos implementar uma difusão confiável em modelos assíncronos, resta-nos levar em consideração que, na prática, estes sistemas não apresentam comportamento indesejado sempre. E desta forma pode-se propor protocolos eficientes para difusão confiável, mesmo com premissas fracas, como as da seção anterior. Existem quatro tipos básicos de protocolos de difusão confiável, cada um deles com seus pontos fortes e fracos, a seguir são apresentados estes protocolos e suas principais características.

3.2.2.1 Protocolos Iniciados pelo Emissor

Os protocolos de difusão iniciados pelo emissor tem como principal característica o fato deste centralizar a coordenação da liberação de mensagens, normalmente, tomando como base mensagens de confirmação (ACKs) enviadas pelos receptores. A liberação da mensagem se dá apenas após a recepção de todos os ACKs (protocolo de duas fases) [28]. A figura 3.5 ilustra o funcionamento deste tipo de protocolo.

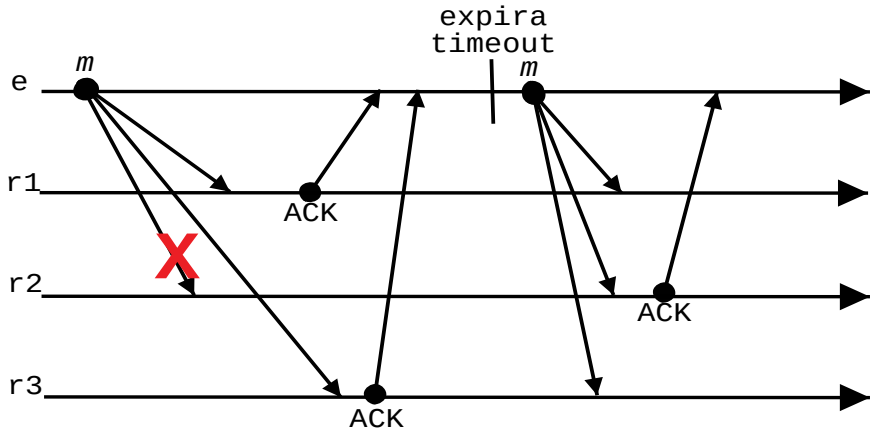

Figura 3.5: Funcionamento de um protocolo iniciado pelo emissor.

Na figura 3.5 vemos que o processo emissor (e), após enviar uma mensagem para um grupo de processos (r_1,r_2 e r_3), recebe ACKs de todos os membros do grupo menos de r_2, o qual não recebeu a mensagem. Nesta ilustração simples, o emissor espera um *timeout* expirar e verifica que não recebeu todos os ACKs do grupo. Assim sendo o mesmo retransmite a mensagem no grupo, como r_1 e r_3 já haviam recebido m, eles ignoram a mesma, já r_2, a recebe e logo em seguida manda um ACK para e, que agora sabe que todo o grupo recebeu m.

Nesta abordagem, a mais simples, o controle sobre os temporizadores e a retransmissão das mensagens fica a cargo do emissor. Entretanto é possível de se implementar um protocolo deste tipo colocando os temporizadores nos receptores: a idéia é que os mesmos enviem periodicamente ACKs

ao emissor e enviem pedidos de retransmissão (não ACKs ou NACKs) se não receberem uma mensagem correta dentro de um determinado período de tempo.

Note que em ambas as abordagens descritas o controle sobre as mensagens em memória que estão disponíveis para serem retransmitidas fica a cargo do emissor. Assim ele apaga uma mensagem enviada da memória se e somente se ele receber ACKs de todos os membros do grupo com relação a esta mensagem.

Apesar de utilizar um esquema simples e intuitivo para prover difusão confiável, os protocolos iniciados pelo emissor sofrem de três graves problemas:

1. O emissor deve conhecer o conjunto de receptores (*membership* ou visão do sistema) e esta informação nem sempre está disponível em sistemas de larga escala;
2. O emissor fica sujeito a uma implosão de ACKs, se o grupo for constituído de muitos membros, o que torna esta solução não escalável;
3. Em caso de perda de algum ACK o emissor fica sem saber se pode liberar a mensagem ou não.

Como exemplo de protocolo de difusão confiável iniciado pelo emissor podemos citar o XTP (*Xpress Transfer Protocol*) [45].

3.2.2.2 Protocolos Iniciados pelo Receptor

Os protocolos iniciados pelo receptor tem como principal característica a não utilização de ACKs. Nestes protocolos os receptores enviam NACKs aos emissores quando detectam que não receberam uma mensagem (de forma integra), requisitando seu reenvio. Esta detecção é feita através de mecanismos de controle de erro (para identificar pacotes corrompidos), números de seqüência ou *timeouts*.

A figura 3.6 ilustra o funcionamento desta classe de protocolos. Neste caso, não existem ACKs, assim, quando r_2 recebe m_2 ele logo percebe que não recebeu m_1[1], e manda um NACK pedindo essa mensagem. Ao receber este NACK, o emissor (e) reenvia m_1 ao grupo, e como na ilustração anterior, r_1 e r_3 a ignoram, enquanto que r_2 a recebe.

Devido ao fato dos emissores receberem notificações apenas quando as mensagens não são recebidas, os mesmos não podem decidir sobre quando uma mensagem pode ser removida do *buffer* de envio (liberada), o que implica que este tipo de protocolo sofre do chamado problema do *buffer* infinito. Este problema afirma que em um protocolo iniciado pelo receptor "puro"é necessário que os emissores tenham memória suficiente para guardar todas as mensagens que enviam, de tal sorte que

[1] Cada mensagem deve vir com um número de sequência relativo a seu emissor.

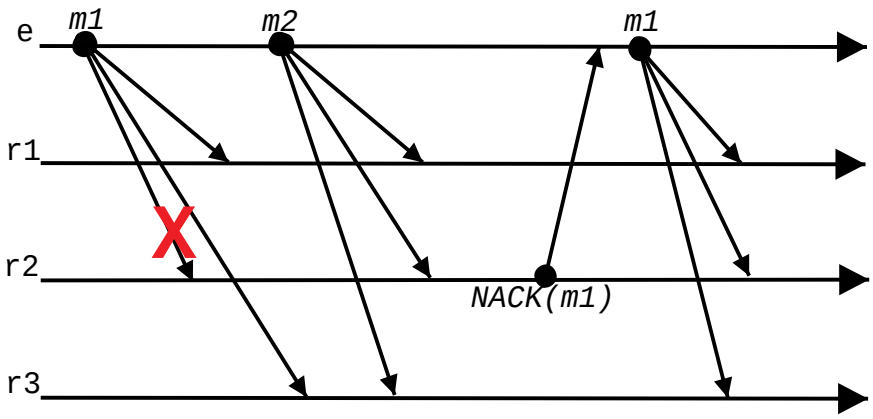

Figura 3.6: Funcionamento de um protocolo iniciado pelo receptor.

se algum receptor enviar um NACK referente a uma mensagem, ela ainda está guardada e pode ser reenviada pelo emissor. Em situações reais este *buffer* teria de ter tamanho infi nito.

Assim como os protocolos iniciados pelos emissores apresentados na seção anterior, os protocolos iniciados pelos receptores também sofrem do problema da implosão de mensagens, só que neste caso, ao invés da implosão de ACKs, acontece a implosão de NACKs. Para resolver este problema, foram criados os protocolos RINA (*Receiver Initiated with NACK Avoidance*) [28].

Um protocolo RINA genérico trabalha da seguinte forma: o emissor difunde a mensagem para o grupo; quando um receptor percebe que perdeu alguma mensagem, espera um tempo aleatório e difunde um NACK para o emissor e todos os membros do grupo. Quando um receptor recebe um NACK de uma mensagem que ele também não recebeu e que ele está esperando para mandar um NACK, ele considera que seu NACK já foi enviado e espera uma resposta do emissor (desta forma ele aproveita o NACK que outro já enviou). Com este esquema espera-se que apenas um NACK seja enviado para o emissor, evitando-se assim a implosão de NACKs. Um esquema que segue este modelo é implementado no protocolo SRM (*Scalable Reliable Multicasting*) [18].

Com base no citado acima, um protocolo iniciado pelo receptor ideal tem três grandes vantagens sobre um protocolo iniciado pelo emissor [28]:

1. O emissor não conhece os membros do grupo, logo não é necessário um serviço de *membership*;
2. O emissor não tem de processar ACKs de cada receptor;
3. O receptor dita o ritmo de envio do emissor, através do envio de NACKs.

A principal limitação deste tipo de protocolo é o fato de não haver um mecanismo que permita ao emissor descartar as mensagens enviadas com segurança.

Como exemplos deste tipo de protocolo podemos citar o LRMP (*Light weight Reliable Multicast Protocol*) [29], o SRM (*Scalable Realiable Multicast*) [18] e o TRM (*Transport protocol for Reliable Multicast*) [42];

3.2.2.3 Protocolos Baseados em Árvore

Esta classe de protocolos procura resolver os problemas apresentados pelos protocolos anteriores organizando o conjunto de receptores em grupos hierárquicos (formando uma árvore de reconhecimento), onde cada grupo local tem um líder que é o responsável pelo recebimento de ACKs dos membros do grupo e pelo envio de um ACK (agregado[2]) a seu líder. Este processo se repete até que o emissor receba os ACKs de seus nós filhos. As figuras 3.7 apresenta um exemplo de uma árvore de reconhecimento.

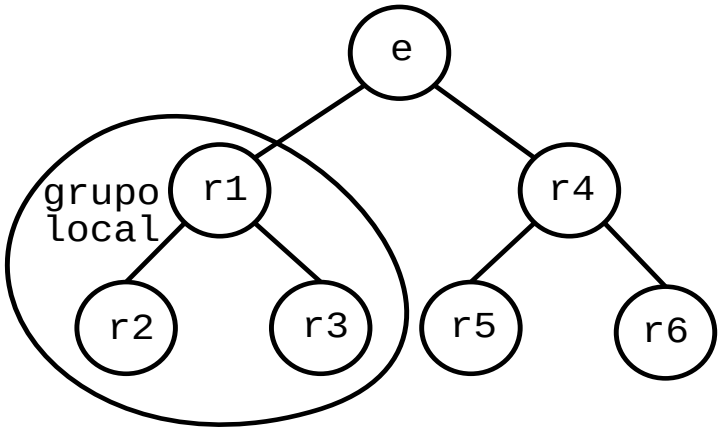

Figura 3.7: Uma árvore de reconhecimento.

Na figura 3.7 o nó r_1 da arvore é o emissor, e os demais nós de seu grupo local são receptores. A figura 3.8 apresenta o funcionamento de um protocolo baseado em árvore para o grupo da figura 3.7.

A comunicação se desenvolve da seguinte forma: o emissor (e) difunde a mensagem para todos os membros do grupo (r_1,r_2,r_3,r_4,r_5,r_6), divididos em grupos locais. Os nós folha da árvore (r_2,r_3,r_5 e r_6) ao receberem a mensagem, enviam ACKs a seus líderes (r_1 e r_4), notificando-os do recebimento da mensagem e estes, por sua vez, ao receberem todos os ACKs de seu grupo enviam um ACK a seu líder (no caso e).

Note que se a árvore de reconhecimento tiver de altura 1 (o emissor e vários nós folhas) temos uma configuração de protocolo iniciado pelo emissor.

[2]Um ACK agraga o ACK do receptor que o enviou e de seus nós filhos.

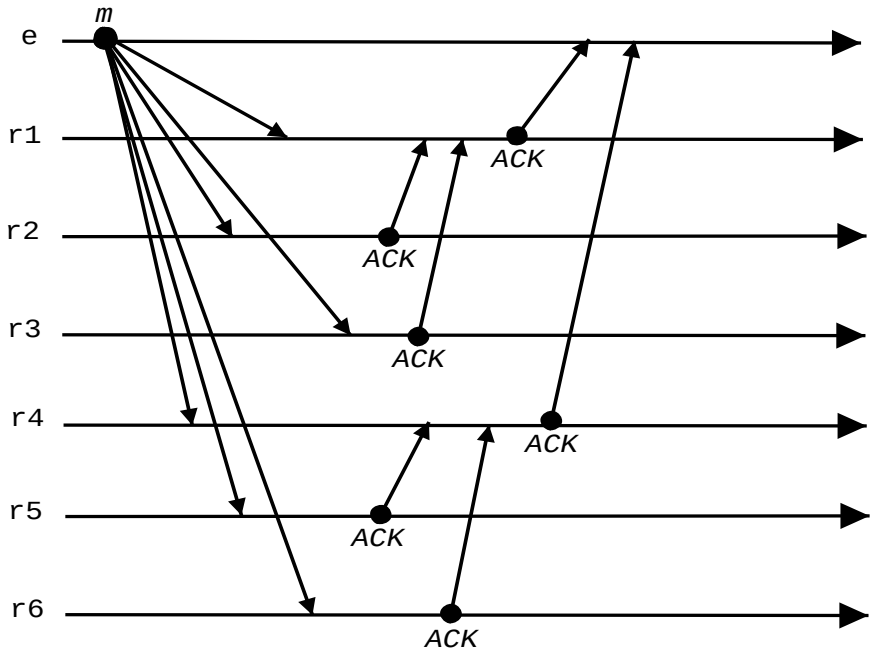

Figura 3.8: Funcionamento de um protocolo baseado em árvore.

O protocolo descrito acima é um modelo clássico que visa organizar a propagação de ACKs hierarquicamente, este modelo é utilizado, entre outros, pelo protocolo RMTP (*Reliable Multicast Transport Protocol*) [30]. Mecanismos adicionais podem ser utilizados para melhorar ainda mais o desempenho e a confiabilidade deste tipo de protocolo; por exemplo, retransmissões locais, onde cada líder tem a responsabilidade de retransmitir as mensagens faltosas para o seu grupo e também liberar as mensagens já recebidas por todos (também utilizada na RMTP). Um outro tipo de protocolo baseado em árvore é o tree-NAPP (*tree-based with NACK Avoidance and Periodic Polling*) [28]. Esta última classe de protocolos é uma extensão hierárquica dos protocolos iniciados pelo receptor. Conforme já apresentado, NACKs sozinhos não são suficientes para garantir confiabilidade. Assim, neste tipo de protocolo, ACKs são enviados periodicamente pelos receptores à seus líderes para que estes saibam qual o número de sequência da ultima mensagem recebida pelos membros de seu grupo - de tal forma que possam descartar mensagens armazenadas no *buffer* de envio (liberando memória e resolvendo o problema do *buffer* infinito).

Os protocolos baseados em árvore eliminam o problema da implosão de ACKs/NACKs e eliminam a necessidade do emissor conhecer todo o conjunto de receptores. Entretanto estes protocolos necessitam de algoritmos para a montagem e manutenção da árvore de reconhecimento para cada emissor, algoritmos estes que muitas vezes aumentam a complexidade do protocolo (vale lembrar que para cada emissor deve haver uma árvore de reconhecimento). Outro problema que protocolos deste tipo enfrentam diz respeito a possíveis falhas que ocorram nos líderes de grupos locais e suas implicações no funcionamento do protocolo.

Além do RMTP, que usa um modelo de árvore de reconhecimento estática[3], outros protocolos estão presentes na literatura. O TMTP (*Tree-based Multicast Transport Protocol*) [51] faz a configuração da árvore de reconhecimento em tempo de execução. O TRAM (*Tree-based ReliAble Multicast protocol*) [11] - parte do JRMS (*Java Reliable Multicast Service*) [25], que é um conjunto de classes e interfaces de um serviço de difusão confiável implementado em Java - também implementa um modelo de árvore dinâmica para o reconhecimento de mensagens.

3.2.2.4 Protocolos Baseados em Anel

Os protocolos baseados em anel para difusão confiável foram inicialmente concebidos para prover suporte a aplicações que requeriam propriedades de ordenação de mensagens. A premissa básica destes protocolos é ter um único membro detentor do *token* responsável pelo envio de ACKs ao emissor. Se um ACK não for recebido do processo *token* (detentor do *token*) antes do *timeout* associado a uma mensagem difundida expirar, esta mensagem deve ser reenviada ao grupo novamente.

Este ACK também serve para marcar *timestamps* (ou número de sequência) das mensagens, assim todos os membros do grupo tem ordenação total de mensagens para a liberação às aplicações. Receptores enviam NACKs ao membro detentor do *token* para que este retransmita os pacotes que foram originalmente enviados pelo emissor. A passagem do *token* de um membro do grupo para outro pode ser realizada de mensagem em mensagem (após cada ACK enviado ao emissor), a cada *N* mensagens ou após determinado período de tempo. O que é importante ressaltar é que o *token* só deve ser passado se houverem garantias de que o novo detentor do mesmo recebeu o mesmo conjunto de mensagens presentes no antigo portador do *token*.

Nossa descrição do funcionamento dos protocolos baseados em anel baseia-se em um dos primeiros protocolos propostos para *multicast* confiável: o TRP (*Token Ring Protocol*) [10] e sua extensão mais moderna para WANs, o RMP (*Reliable Multicast Protocol*) [28], onde a principal diferença para o anterior diz respeito a uma hierarquia de anéis utilizada para garantir a confiabilidade do *multicast* em redes de larga escala.

Um protocolo baseado em anel um pouco diferente é utilizado no sistema de comunicação de grupo Totem [38]. Este sistema utiliza-se de redes baseadas em anéis lógicos para prover um mecanismo de comunicação de grupo confiável com ordenação total. Sendo assim, uma hierarquia de anéis semelhante a do RMP é utilizada para formar anéis de LANs que se comunicam. Este sistema também oferece garantias de entrega e ordenção total de mensagens. O protocolo utilizado pelo Totem baseia-se na idéia de que o membro do grupo de posse do *token* é o único com direito de enviar mensagens. Quando este membro envia uma mensagem ele incrementa o campo `seq` presente no *token* e coloca este novo valor de `seq` na mensagem. Os outros membros do grupo detectam mensagens faltando a partir de falhas na sequência das mensagens recebidas. Quando um membro detecta

[3] A configuração da árvore é definida antes do ínicio da transmissão

que lhe falta uma mensagem, este espera que o *token* fique em seu poder e então coloca o número da mensagem que lhe falta no campo `rrl` (*retrasmission request list*) do *token* e o passa, desta forma, quando outro membro do anel pegá-lo, ele saberá que alguém não tém as mensagens marcadas no `rrl`, e se ele as tiver, as retransmitirá retirando a marcação do `rrl` no *token*. A figura 3.9 apresenta esta idéia.

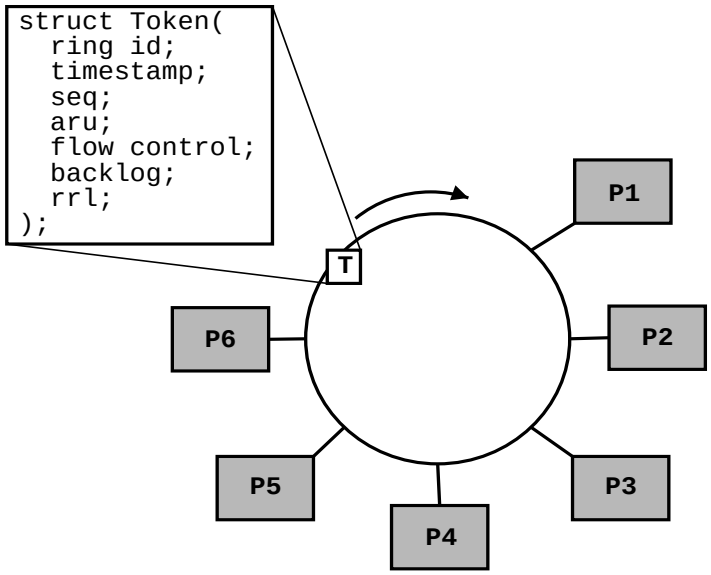

Figura 3.9: Um anel local no Totem.

Na figura 3.9 é possível ver que o *token* é apenas uma estrutura de dados que mantém o estado da comunicação no anel, guardando informações como o número da ultima mensagem enviada e quais mensagens devem ser retransmitidas, entre outras.

Assim como os protocolos baseados em árvore, os protocolos baseados em anel organizam os membros do grupo de tal forma a evitar a implosão de ACKs/NACKs e garantir a confiabilidade do *multicast* mesmo com memória finita. Entretanto, assim como os anteriores, esta classe de protocolos apresenta vários problemas, que dizem respeito principalmente a configuração do anel, tanto inicialmente quanto a medida que o grupo se altera, e a presença de falhas parciais que resultam na perda do *token*. Estes casos devem ser tratados cuidadosamente neste tipo de protocolo, o que resulta geralmente em um acréscimo de complexidade considerável de suas implementações.

3.2.3 Análise Comparativa

A tabela 3.3 apresenta um resumo comparativo dos principais tipos de protocolos de difusão confiável baseados em premissas, de serviços não confiáveis subjacentes, disponíveis na literatura.

Nesta tabela temos um resumo das principais características dos protocolos apresentados nesta

Tipo de Protocolo	*Membership*	*Buffer* Infinito	Organização	Ordem de Mensagens
Iniciado pelo Emissor	sim	não	ndef	$O(R \ln R)$
Iniciado pelo Receptor	não	sim	ndef	$O(\ln R)$
Baseado em Árvore	sim	não	árvore	$O(1)$
Baseado em Anel	sim	sim	anel	$O(R)$

Tabela 3.3: Tipos de Protocolos e suas Características.

seção. A tabela mostra quais os protocolos que exigem lista de membros dos grupos (*membership*), quais necessitam de *buffer* infinito para garantir a integridade da comunicação, qual a organização dos membros dos grupos nesse protocolo e a complexidade medida em mensagens trocadas no grupo considerando-se R receptores (membros do grupo) e uma taxa de perda de mensagens constante e não nula [28].

3.3 Conclusão

Neste capítulo foram apresentados o *multicast* IP como principal tecnologia de comunicação de grupo em redes de larga escala (em especial a Internet) e os principais tipos de protocolos que implementam comunicação confiável sobre esta tecnologia.

As primitivas de comunicação do *multicast* IP são caracterizadas por um serviço multiponto não confiável e pela ausência de propriedades que reforcem o serviço. Estas limitações tornam praticamente impossível garantir as propriedades necessárias para suportes de grupo sobre esta tecnologia; principalmente se considerarmos as condições de sistemas assíncronos presentes na rede mundial.

Partindo das características de sistemas reais que mesmo sobre a rede mundial, apresentam intervalos de comportamento síncrono, apresentamos neste capítulo um conjunto de protocolos que fazem ou podem fazer uso destas primitivas não confiáveis, no fornecimento de serviços apresentando melhores condições de confiabilidade.

As considerações apresentadas neste capítulo são a base para a definição de um protocolo de difusão confiável iniciado pelo receptor que acrescenta confiabilidade ao modelo de comunicação de grupo não confiável do CORBA apresentado no capítulo 4.

Capítulo 4

Comunicação de Grupo no CORBA

O padrão CORBA, *Common Object Request Broker Architecture* [21], define uma arquitetura de *middleware* para o conceito de objetos distribuídos. Esta arquitetura permite que objetos instanciados em diferentes *hosts* da rede se comuniquem de maneira transparente através de chamadas de métodos remotos (RMI). A arquitetura CORBA foi proposta como uma solução para a programação de sistemas distribuídos heterogêneos. Esta arquitetura enfatiza o modelo de interações cliente/servidor e padrões abertos.

Embora a proposição da OMG (*Object Management Group*) apresente uma boa solução para comunicação ponto a ponto, algumas aplicações distribuídas trabalham sobre a idéia de grupos de objetos, em especial as que implementam algum suporte a tolerância a faltas. Estas aplicações necessitam de algum tipo de suporte de comunicação de grupo.

Assim nos últimos anos a OMG vem lançando algumas especificações cuja finalidade é a introdução da noção de grupos de objetos dentro da arquitetura CORBA. Dentre estas especificações destacam-se a *Fault Tolerant* CORBA [20] e a *Unreliable Multicast Inter-ORB Protocol* [22]. A primeira define uma série de serviços que dão suporte a objetos tolerantes a faltas (através de grupos de objetos replicados) e a segunda define um mapeamento das chamadas de métodos CORBA para o *multicast* IP.

Este capítulo apresenta uma pequena introdução da arquitetura CORBA e seus principais componentes e também uma descrição dos dois esforços relacionados à integração de modelos de grupos em sua arquitetura. Ao final são apresentadas algumas considerações a respeito da integração do FT-CORBA com o UMIOP e uma proposta de arquitetura integrada que forneça replicação ativa no FT-CORBA utilizando o UMIOP como suporte de comunicação de grupo.

4.1 Arquitetura CORBA

Com o advento da Internet, sistemas distribuídos inicialmente baseados em redes locais (ambientes normalmente homogêneos), tomaram as dimensões de sistemas de larga escala que são executados em ambientes heterogêneos em termos de *hardware*, linguagens de programação e tecnologias de rede. Esta transformação levou a introdução de novos padrões e muitas tecnologias de *middleware* para o suporte à programação distribuída. Reconhecendo a urgência do problema, em abril de 1989 um conjunto de empresas e instituições de pesquisas formaram a OMG. A OMG teve desde o seu início como principal objetivo a disseminação da tecnologia de objetos distribuídos.

Em outubro de 1991 a OMG lançou o padrão CORBA, que introduzia uma infra-estrutura de objetos distribuídos completamente independente de plataforma. Neste padrão, objetos possuem suas interfaces definidas em uma linguagem de definição de interfaces (IDL) independente de linguagem de programação. Estas interfaces podem ser exportadas para a tecnologia de implementação do sistema.

Esta seção apresenta um pequeno resumo dos principais elementos da arquitetura CORBA.

4.1.1 O Modelo de Comunicação CORBA

A arquitetura CORBA está centrada em dois elementos básicos: o ORB (*Object Request Broker*) e a IDL (*Interface Definition Language*). O ORB é o barramento de objetos responsável pela troca de mensagens entre componentes em um sistema CORBA. A IDL é a linguagem utilizada na definição das interfaces dos objetos, ela se faz presente para garantir que as funcionalidades exportadas por um objeto CORBA sejam independentes da linguagem de programação em que este é implementado.

A figura 4.1 apresenta o modelo básico de comunicação do CORBA, que implementa um RMI. Nesta figura, temos uma requisição passando de um cliente para uma implementação de objeto remoto (objeto CORBA).

A partir da figura 4.1, dois aspectos podem ser levantados [44]:

1. Tanto o cliente quanto a implementação do objeto remoto estão isolados do ORB pela interface independente de linguagem especificada inicialmente em IDL;
2. A requisição não passa diretamente do cliente para a implementação do objeto. Em um sistema CORBA, como na maioria das implementações de RMI, a execução de requisições dependem de suporte do ORB.

O mecanismo de RMI do CORBA compreende basicamente a transformação da invocação de método em uma mensagem e sua transmissão ao *host* destino através do serviço de *sockets* fornecido

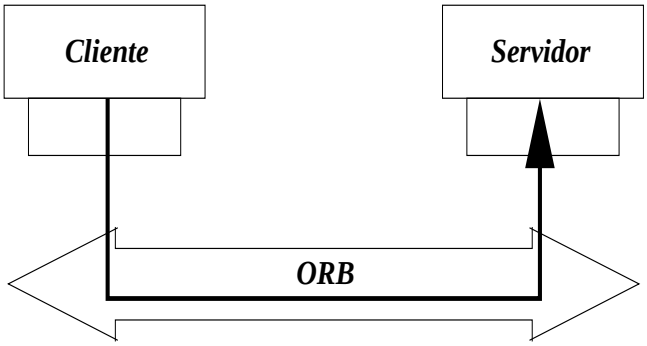

Figura 4.1: Modelo de comunicação básico do CORBA.

pelo sistema operacional e protocolos de transporte subjacentes, e ainda, a sua recuperação no *host* destino com a transformação inversa e a invocação do método na implementação do objeto.

A figura 4.2 apresenta como é realizada a conversão de invocação de método para mensagens e vice-versa.

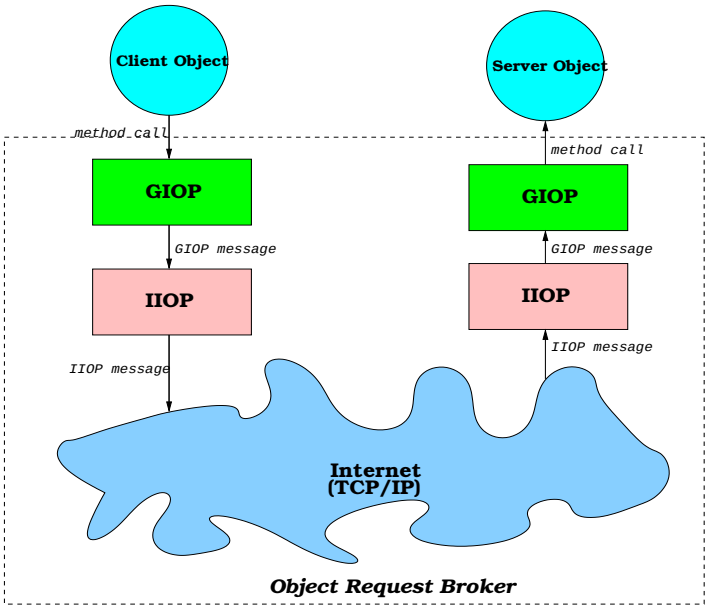

Figura 4.2: Funcionamento (simplificado) do CORBA com IIOP.

Na figura 4.2, é apresentado o protocolo GIOP, *General Inter-ORB Protocol*, que como o próprio nome sugere é um protocolo genérico que define mensagens e formatos para a interoperabilidade entre ORBs diferentes. A sintaxe de transferência definida no GIOP (CDR - *Common Data Representation*) é a base para comunicações na arquitetura CORBA; todas as requisições quando transmitidas são codificadas nesta sintaxe na forma de seqüências de bytes. O protocolo GIOP é genérico e quando mapeado em protocolos de transporte define protocolos específicos de mapeamento. Para a Internet,

por exemplo, um destes protocolos específicos é o IIOP (*Internet Inter-ORB Protocol*) que é uma concretização do GIOP considerando o protocolo de transporte TCP.

O IIOP foi definido para comunicação entre um emissor e um receptor, conforme apresentado na figura 4.2, como não poderia ser diferente, dadas as características do protocolo de transporte que ele utiliza (TCP). Outra característica deste protocolo é a confiabilidade. O TCP garante que se não houverem falhas nos *hosts* comunicantes e no canal de comunicação, todas as mensagens serão entregues com ordenação FIFO.

4.1.2 Visão Geral da Arquitetura CORBA

A figura 4.3 mostra a arquitetura de um ambiente CORBA [21]:

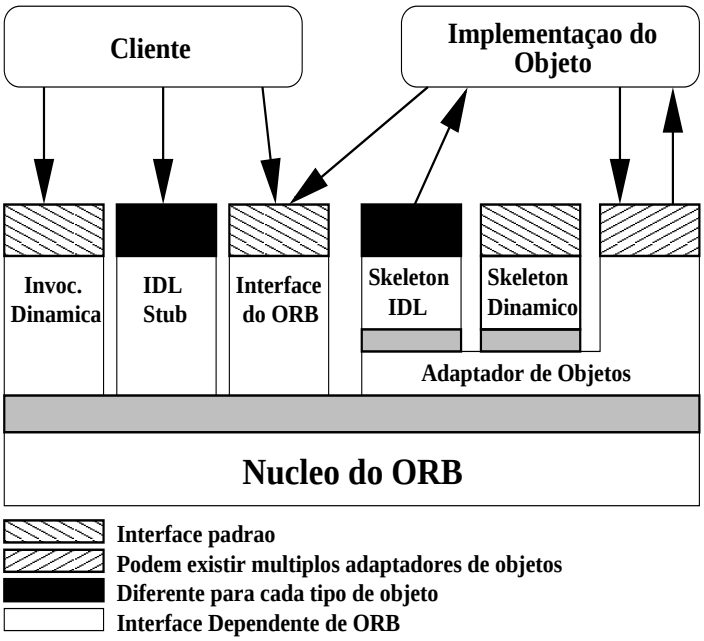

Figura 4.3: Arquitetura CORBA.

O mecanismo de RMI suportado pelo CORBA é gerado, em sua versão estática, a partir da tradução da especificação IDL da interface do objeto servidor, dando origem aos *stubs* e *skeletons* IDL que implementam grande parte dos aspectos de serialização e deserialização (*marshaling* e *unmarshaling*) de requisições remotas. Interfaces dinâmicas para RMI são fornecidas também pelo CORBA, o que permite que chamadas dinâmicas sejam montadas (DII - *Dynamic Invocation Interfaces*) e objetos sem *skeletons* compilados sejam localizados e ativados (DSI - *Dynamic Skeleton Interface*). Estas interfaces dinâmicas permitem uma certa flexibilidade em tempo de execução no modelo de objetos distribuídos do CORBA.

A linguagem IDL, um dos componentes centrais da arquitetura CORBA na garantia de interoperalidade, é um sub-conjunto da linguagem C++ acrescida de algumas palavras-chave que permitem a descrição de interfaces remotas e outros elementos relacionados à objetos distribuídos. A IDL é uma linguagem declarativa (não algoritimica).

Os objetos CORBA podem ser acessados, independente de suas linguagens de implementação, uma vez definidas suas interfaces em IDL. A tradução destas interfaces dão origem aos *stubs* e *skeletons* nas linguagens de implementação do cliente e servidor respectivamente.

Do ponto de vista operacional, um cliente consegue acessar uma implementação de um objeto CORBA através da referências deste, chamada IOR (*Interoperable Object Reference*). A IOR contém um conjunto de informações relacionadas ao mecanismo de transporte utilizado pelo ORB, que são encapsuladas em perfis. Por exemplo, a figura 4.4 apresenta um perfil IIOP, utilizado pelo núcleo do ORB para realizar comunicações através do TCP/IP.

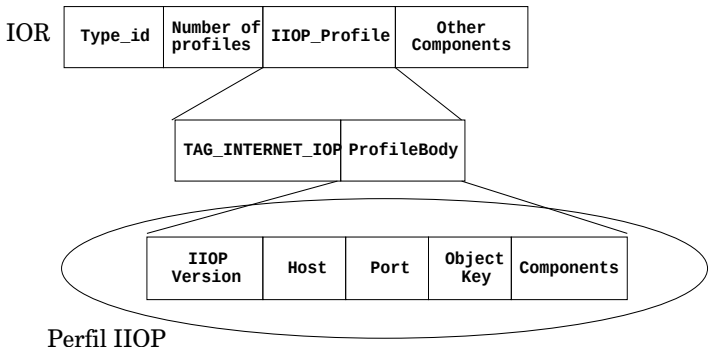

Figura 4.4: Referência de Objetos com Perfil IIOP

O Adaptador de Objetos, também presente na figura 4.3, é o responsável pelas interações entre o mundo dos objetos CORBA e o mundo das implementações em determinada linguagem de programação. Assim, sua principal função é localizar, ativar e encaminhar mensagens aos *skeletons* (e as implementações associadas) a partir de tabelas que traduzem endereços locais a partir de referências à objetos. O adaptador de objetos padrão definido nas especificações CORBA é o POA (*Portable Object Adapter*).

4.2 *Fault-Tolerant* CORBA

A OMG começou a se preocupar com a introdução do suporte a tolerância a faltas no CORBA a partir de 1998 com a publicação de uma RFP (*Request For Proposals*) pedindo propostas de serviços relacionados a esta funcionalidade. Esta RFP define ainda uma série de requisitos a serem atendidos por estas propostas.

Como resultado deste RFP e das propostas recebidas pela OMG chegou-se a especificação do padrão FT-CORBA [20]. Através deste padrão a tolerância a faltas é obtida no CORBA através de técnicas de replicação de objetos, baseadas em mecanismos de detecção e recuperação de faltas.

O padrão criado especifica uma série de objetos de serviços usados no fornecimento da tolerância a faltas em objetos CORBA. O padrão fornece tolerância a faltas em hipóteses de faltas de parada. A especificação define ainda quatro tipos de replicação para os grupos de objetos mantidos pelo FT-CORBA:

- **Sem Estado:** Estilo de replicação em que o estado dos objetos não é alterado pelas requisições. Neste caso, quando ocorrem falhas de objetos, não existem transferências de estados durante a recuperação do sistema;
- **Passiva Fria:** Apenas uma réplica do grupo (objeto ou réplica primária) executa as requisições do cliente. As demais réplicas (secundárias) ficam em modo "*standby*"(reserva). O primário grava periodicamente informações de estado (*checkpoint*). Em caso de falha do objeto primário uma das réplicas secundárias assume o papel de novo primário, atualizando seu estado a partir do último *checkpoint* gravado e das mensagens difundidas no grupo desde então;
- **Passiva Morna:** Este modo funciona de maneira semelhante ao anterior com a diferença de que os *checkpoints* feitos pela réplica primária são difundidos para as réplicas secundárias, agilizando desta forma a atualização de uma réplica em caso de falha;
- **Ativa:** Todos as réplicas executam as requisições. Este estilo de replicação segue o modelo de máquinas de estados definido em [43].

Dentre estes quatro tipos de replicação a única que é prevista, mas não especificada ainda, é a replicação ativa, que deverá aparecer em futuras versões do FT-CORBA.

Independente do tipo de replicação utilizado, as especificações FT-CORBA permitem que ferramentas proprietárias de comunicação de grupo possam ser usadas para que requisições possam ser difundidas (de preferência através de difusão atômica) por todos os membros do grupo.

Esta seção apresenta a arquitetura básica do padrão FT-CORBA e os mecanismos que garantem a interoperabilidade deste padrão com o restante da arquitetura CORBA.

4.2.1 Arquitetura FT-CORBA

A arquitetura FT-CORBA apresenta os serviços, em nível de *middleware*, responsáveis por prover as funcionalidades básicas para aplicações tolerantes a faltas. Esses serviços estão divididos em três módulos básicos:

- **Gerenciamento de Replicação (SGR)**: Este é o serviço responsável pelo ciclo de vida dos grupos. Ele oferece dois tipos de funcionalidades: gerenciamento de propriedades dos grupos e o gerenciamento de grupos, com criação de membros (através de Fábricas de Objetos) e o controle de *membership* dos grupos definidos;

- **Gerenciamento de Falhas (SGF)**: É o serviço responsável pela detecção, notificação, análise e diagnose de falhas. Este serviço trabalha em conjunto com o SGR para que este último mantenha um *membership* sempre atualizado;

- **Gerenciamento de Recuperação e *Logging* (SRL)**: O SRL é responsável pela consistência de estados das réplicas. Este serviço armazena as requisições enviadas ao grupo e realiza a atualizações nos membros através de *checkpoints*. Além disso também é responsabilidade do SRL atuar na recuperação de réplicas faltosas para atualização de seus estados.

Estes três módulos definem vários objetos de serviço. A figura 4.5 apresenta a arquitetura FT-CORBA com estes objetos de serviço.

No *host* 1 não existem intervenções dos serviços FT-CORBA. Assim para o cliente, a tolerância a faltas do sistema é completamente transparente, considerando que seu ORB utilize o sistema de comunicação de grupo para difundir as mensagens entre os membros do grupo.

A criação de membros do grupo é feita através do gerenciador de replicação (SGR) que delega esta funcionalidade chamando o método `create_object()` na fábrica do *host* correspondente. Este objeto fábrica (SGR) por sua vez é responsável localmente pela criação e ativação do novo membro em seu *host*. Os *hosts* 2 e 3, da figura 4.5, contém membros do grupo servidor de objetos replicados criados a partir das fábricas ativas nestes.

A detecção de falhas em réplicas ocorre da maneira hierárquica, de acordo com três níveis de abstrações: objeto, processo e *host*. Em cada processo que mantém membros do grupo existe um detector de falhas de objeto (SGF). Este detector verifica periodicamente (através da chamada de método `is_alive()`) se os objetos ativos em seu processo ainda estão em atividade. Em nível de *host* também existe um detector de falhas (detector de falhas de processo) que faz o monitoramento dos processos. Estes detectores, em nível de *host*, estão subordinados ainda, aos detectores em nível global, responsáveis pela detecção de falhas em *hosts*. Os detectores de falhas de *host* enviam, periodicamente, requisições aos detectores de falhas de processo (ativando também o método `is_alive()` destes), para verificar se estes ainda estão ativos.

Quando qualquer um dos detectores presentes no sistema observam uma parada em um objeto, processo ou *host*, esta é reportada ao notificador de falhas, que as repassa ao serviço de replicação para que este atualize o *membership* do grupo. Se a falha ocorreu no membro primário do grupo cabe ao mecanismo de recuperação (de acordo com o estilo de replicação escolhido para este grupo)

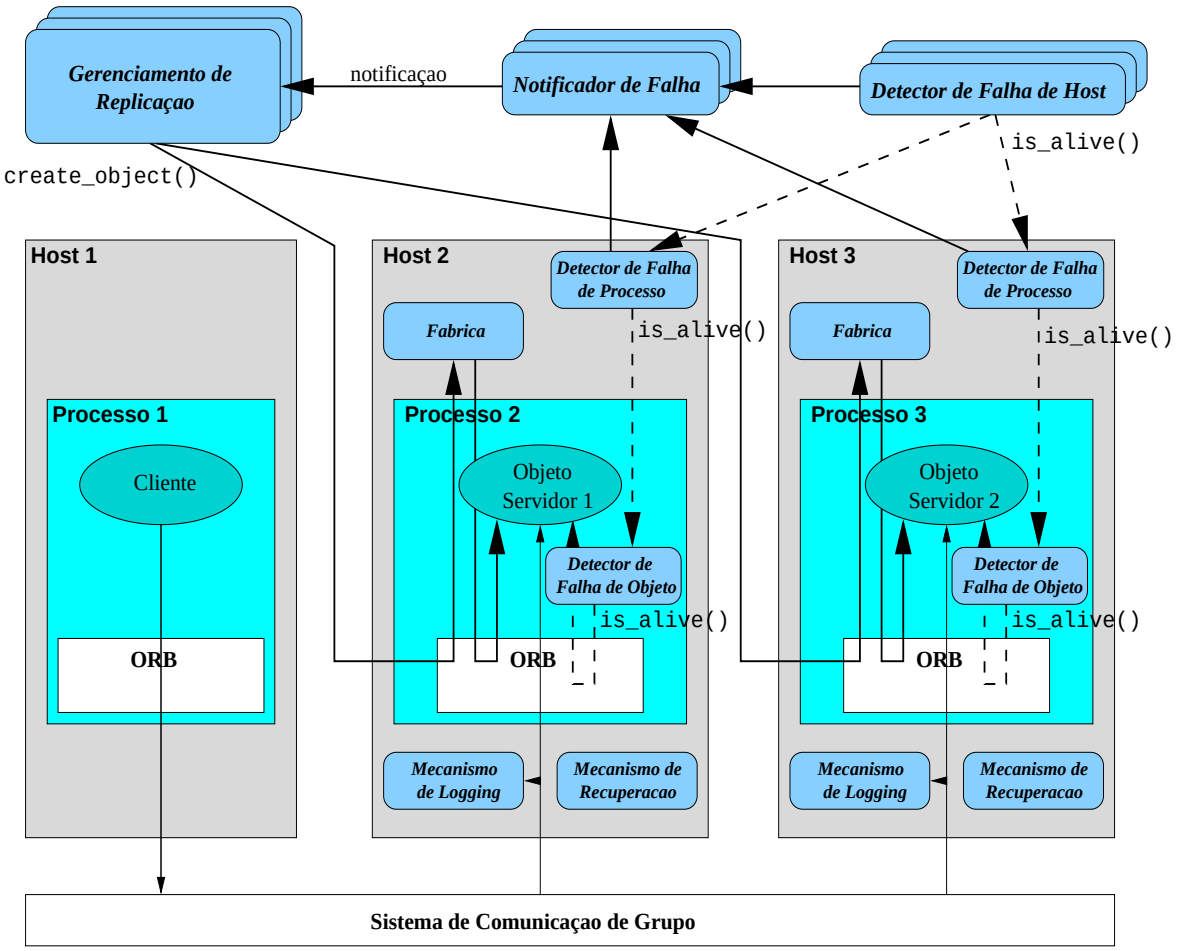

Figura 4.5: Arquitetura FT-CORBA.

escolher um novo primário e atualizar seu estado de tal forma que este possa continuar processando requisições para o grupo do ponto onde o objeto faltoso parou.

A fim de obter uma melhor confiabilidade dos sistemas baseados no FT-CORBA, os objetos de serviço globais (gerenciador de replicação, notificador de falhas e detector de falhas de *host*) devem ser replicados.

4.2.2 Interoperabilidade do FT-CORBA

Na maioria de suas especificações, a OMG define apenas interfaces (em IDL) de serviços e a semântica associada a cada uma das operações definidas nestas interfaces. No caso do FT-CORBA não é diferente, a OMG define interfaces para todos os serviços presentes na arquitetura, com excessão dos mecanismos de *logging* e recuperação, que não são acessados pelos objetos de aplicação.

Conforme já discutido, as especificações FT-CORBA não define um suporte de comunicação de

grupo: não existem interfaces e nem tão pouco um protocolo padronizado para este importante módulo. As especificações apenas sugerem que se utilize qualquer suporte proprietário de comunicação de grupo.

A fim de manter a interoperabilidade entre grupos de objetos, de maneira independente do suporte de grupo adotado pelos ORBs que implementam o FT-CORBA, a OMG criou um formato padronizado para a referência de grupo de objetos: IOGR (*Interoperable Object Group Reference*). O formato do IOGR é apresentado na figura 4.6.

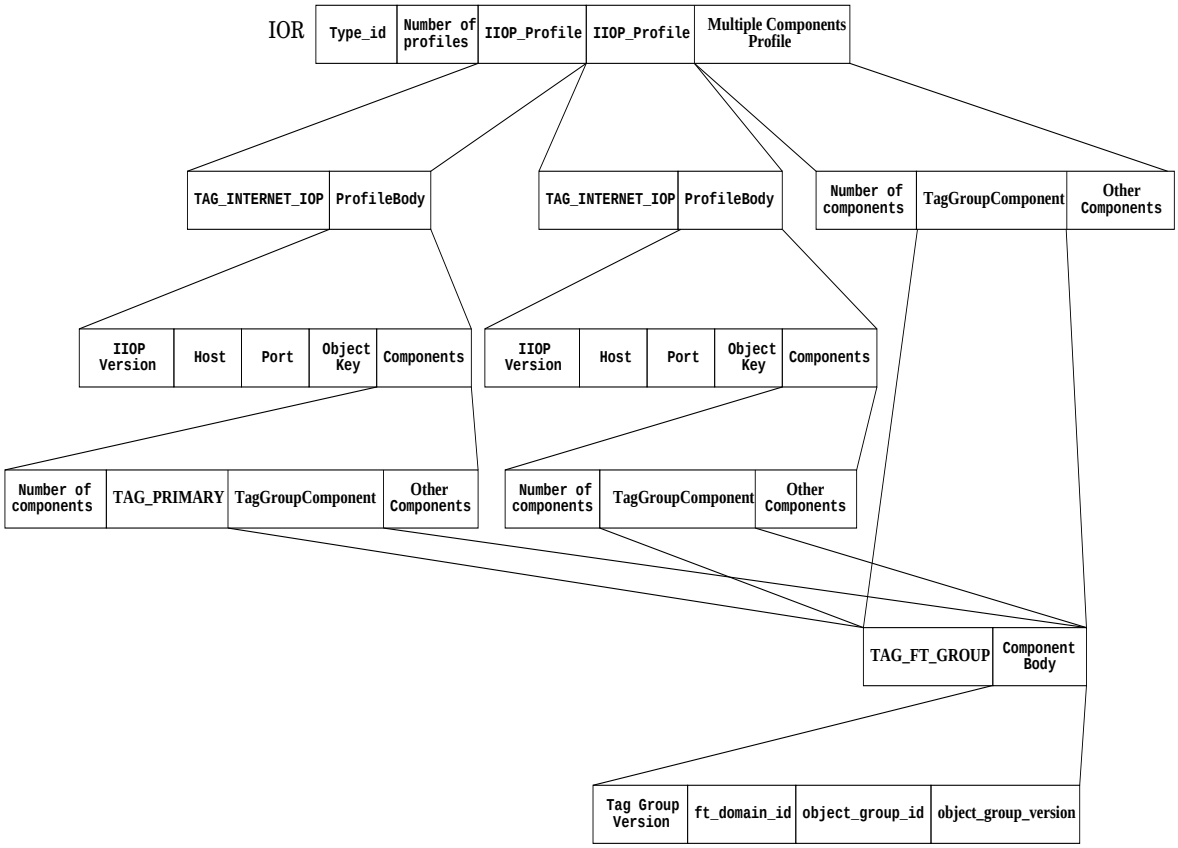

Figura 4.6: Estrutura da IOGR.

A estrutura apresentada na figura 4.6 é bastante semelhante ao da referência de objeto simples (IOR). A grande diferença entre estes dois tipos de referências reside no fato de que ao invés de ter apenas o perfil relacionado ao objeto correspondente a referência, a IOGR contém uma série de perfis, um para cada membro do grupo. Também há uma estrutura chamada `TagGroupComponent` que contém informações a respeito do grupo em si, como seu identificador, domínio e versão.

O membro primário do grupo é definido a partir da presença da *tag* `TAG_PRIMARY` na lista de componentes de seu perfil.

4.3 *Unreliable Multicast Inter-ORB Protocol*

Em 1999 a OMG iniciou o processo de especificação de um protocolo de difusão não confiável baseado em *multicast* IP e um modelo de grupo de objetos que desse suporte a este protocolo em ORBs CORBA. Este processo culminou com o lançamento das especificações UMIOP (*Unreliable Multicast Inter-ORB Protocol*) [22], que serão apresentadas nesta seção. Primeiramente será discutido o mapeamento do GIOP na pilha UDP/*multicast* IP; depois serão analisados os aspectos referentes ao protocolo MIOP (*Multicast Inter-ORB Protocol*), que é a parte da especificação que orienta este mapeamento, fornecendo então comunicação de grupo não confiável entre ORBs. Na seqüência é discutido o papel do MGM (*Multicast Group Manager*), um componente opcional da especificação que encapsula uma série de funcionalidades para a criação e gerenciamento de grupos na forma de um objeto de serviço.

4.3.1 O protocolo MIOP

O objetivo do MIOP é prover um mecanismo para a entrega de mensagens GIOP via *multicast*. O mecanismo de transporte padrão do MIOP é o UDP [39] que tem seus datagramas transferidos fazendo uso do *multicast* IP. Ao utilizar o UDP, protocolo sem conexão e sem garantia de entrega, como protocolo de transporte, o acesso aos serviços do *multicast* IP é conseguido praticamente sem nenhum *overhead*.

Uma mensagem GIOP transmitida via MIOP, é encapsulada em um conjunto de pacotes MIOP. Um pacote MIOP é composto de um cabeçalho e um bloco de dados GIOP.

O tamanho máximo do bloco de dados GIOP que pode ser carregado em um único pacote MIOP depende do tamanho do *frame* suportado pelo hardware de rede utilizado. A *Ethernet*, por exemplo, suporta frames de 1518 bytes. Entretanto vale lembrar que o protocolo UDP permite mensagens de 512 a 65536 bytes.

A figura 4.7 apresenta a definição em IDL do cabeçalho do pacote MIOP.

O papel do cabeçalho MIOP, definido pela estrutura `PacketHeader_1_0`, é permitir a reconstrução de mensagens GIOP no ORB receptor. Os campos desta estrutura são descritos a seguir:

- `magic`: Este campo identifica que a mensagem é um pacote MIOP. Seu valor deve ser sempre o literal `'MIOP'`.
- `hdr_version`: Este campo contém a versão do cabeçalho. Os quatro bits de maior ordem correspondem à versão maior e os quatro de menor ordem à versão menor. O valor deste campo nesta versão da especificação deve ser `10H`.

```
module MIOP
{
  typedef sequence<octet,252> UniqueId;

  struct PacketHeader_1_0
  {
    char               magic[4];          //4 bytes
    octet              hdr_version;       //1 byte
    octet              flags;             //1 byte
    unsigned short     packet_length;     //2 bytes
    unsigned long      packet_number;     //4 bytes
    unsigned long      number_of_packets; //4 bytes
    UniqueId           id;                //1-252 bytes
  };
};
```

Figura 4.7: IDL para o cabeçalho MIOP.

- `flags`: Este campo contém alguns *flags*, definidos dos bits de menor para os de maior ordem, a saber: codificação (*big*(0) ou *little endian*(1)) e fim de mensagem (1 para o último pacote de uma coleção). Os outros seis bits não são utilizados na versão atual do MIOP, e devem conter o valor `0`.
- `packet_length`: Contém o tamanho do bloco de dados contido no pacote. Todos os pacotes de uma coleção, com excessão do último, devem ter o mesmo valor neste campo.
- `packet_number`: O número do pacote em sua coleção. Por exemplo, se uma mensagem GIOP é transformada em 20 pacotes MIOP, este campo deve ter o valor `0` no primeiro pacote e `19` no último.
- `number_of_packets`: Este campo é opcional, entretanto ele pode ser usado, em especial no primeiro pacote de uma coleção, para informar ao receptor quantos pacotes devem chegar para completar esta mensagem. Quando este campo não é utilizado, seu valor deve ser `0`.
- `UniqueId`: Campo que identifica de maneira única uma mensagem. O tamanho máximo deste campo é 252 bytes, entretanto ele deve ser o menor possível. O único requisito associado a este campo é que cada coleção de pacotes deve ter um identificador único, e os pacotes de uma mesma coleção devem ter o mesmo identificador (que é associado ao pacote).

Após o cabeçalho, deve se seguir um "demarcador"de 8 bytes antes de começarem os bytes do fragmento de dados GIOP.

Conforme já discutido, o MIOP não é um protocolo confiável, assim a possibilidade da perda de pacotes existe. Se algum pacote de uma coleção não for recebido, mecanismos usuais como

temporizadores devem ser utilizados para aguardar até que a mensagem (coleção de pacotes) esteja completa. Se algum pacote da coleção não chegar, a coleção (e por conseguinte a mensagem) deve ser descartada.

4.3.2 Modelo de Objetos UMIOP e Suporte a Grupos

O modelo convencional de objetos CORBA especifica que uma referência de objeto deve corresponder a uma única implementação do objeto via uma chave de objeto (*Object Key*). Além disso, a semântica de invocação do CORBA ponto a ponto é confiável em relação a entrega e ordenação de mensagens que podem ser definidas com ou sem espera de resposta.

O MIOP não define um identificador de objetos, mas sim um identificador de grupo que pode ser associado a múltiplos identificadores de objetos (*Object Id*), que são utilizados pelo POA para a ativação das implementações correspondentes [22]. A semântica de entrega de mensagens do UMIOP é sem garantia e controle de erros e ainda sem suporte a respostas. Como não existem requisitos de confiabilidade, não é necessário um serviço de *membership* no sentido de se ter conhecimento de quantos e quem são os membros do grupo. Assim é possível que uma aplicação fique enviando mensagens sem que alguém as esteja recebendo.

Um grupo de objetos no UMIOP consiste em informações de identificação do grupo (um identificador único para este) e informações sobre como acessá-lo na rede de comunicações (endereço IP classe D e uma porta). A figura 4.8 representa (de maneira bastante simplificada) o funcionamento do CORBA com o protocolo MIOP. Nesta figura, um objeto cliente envia uma mensagem via MIOP a um grupo de objetos.

4.3.2.1 Perfil para Difusão Não Confiável

O perfil para difusão não confiável, que representa as informações de transporte do grupo, é definido, na especificação UMIOP, pela estrutura `UIPMC_ProfileBody`. Esta estrutura contém todas as informações requeridas para realização de uma invocação a uma implementação utilizando UDP/*multicast* IP como pilha de protocolos de comunicação. Este perfil difere do tradicional perfil IIOP nos seguintes pontos:

- Não existe chave de objetos;
- O campo que continha o nome do *host* onde o objeto está implementado foi substituído por um campo que deve conter um endereço IPv4/v6 de *multicast*, ou um alias para este endereço.

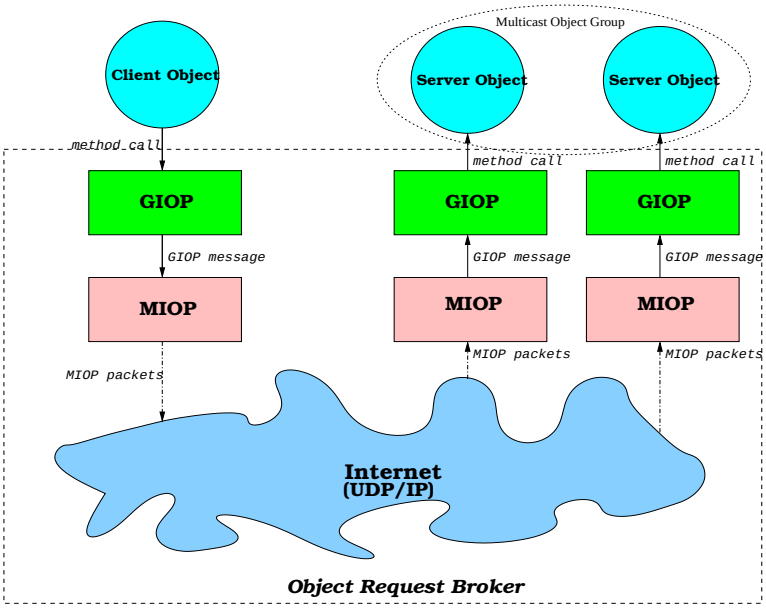

Figura 4.8: Funcionamento (simplificado) do CORBA com MIOP.

O falta de um identificador de objetos neste perfil se deve ao fato das requisições MIOP tipicamente terem mais de um receptor e estes poderem estar em ORBs diferentes. Este cenário é bem mais complexo do que o dos perfis IIOP, que definem um identificador para uma implementação.

A definição em IDL do `UIPMC_ProfileBody` é apresentada na figura 4.9.

A seguir os campos do perfil definido na figura 4.9 são detalhados:

- `miop_version`: Este campo contém a versão do perfil. A versão atual é 1.0. A versão definida neste campo não tem ligação com a versão definida no cabeçalho dos pacotes MIOP (que é a versão do cabeçalho);
- `the_address`: Este campo especifica um endereço IP de *multicast* válido ou um alias para um endereço deste;
- `the_port`: A porta associada ao endereço destino que identifica os processos receptores;
- `components`: Lista de componentes auxiliares. Um componente que deve sempre estar presente nesta lista é a estrutura `TagGroupComponent`, que identifica um grupo de maneira única. Outro componente usualmente utilizado é o perfil IIOP de grupo, que é utilizado em operações com resposta[1].

[1]Como o perfil UIPMC só suporta mensagens *oneway* (sem resposta), faz-se necessário um perfil IIOP que responda as requisições *two-way* implícitas a objetos CORBA, como o método `is_nil()` definido para todos os objetos remotos.

```
module MIOP
{
  ...
  typedef GIOP::Version Version;
  typedef string Address;

  struct UIPMC_ProfileBody
  {
    Version     miop_version;
    Address     the_Address;
    short       the_port;
    sequence<IOP::TaggedComponents> components;
  };
};
```

Figura 4.9: IDL para o `UIPMC_ProfileBody`.

Vale lembrar que a estrutura `UIPMC_ProfileBody` esta sempre presente em referências de grupos UMIOP. Este tipo de referência é apresentado na próxima seção.

4.3.2.2 Referência de Grupo UMIOP

Conforme já apresentado, a IOR serve como identificação única de um objeto (implementação) no CORBA. Entre os seus campos temos o endereço da implementação (tipicamente um IP e porta) e um número que a identifica de maneira única no ORB servidor (chave de objeto). Este IOR é utilizado pelos ORBs clientes como ponteiros para as implementações, endereçando as requisições de métodos à elas através dele.

Uma IOR de grupo serve basicamente para realização de três tipos de invocação:

- A um *gateway* IIOP, nos casos em que o cliente, através de seu ORB, não é capaz de difundir mensagens via *multicast* IP;
- A servidores que suportam MIOP, implementam a mesma interface e pertencem ao mesmo grupo;
- A um servidor IIOP do grupo que deve responder às requisições de operações *two-way* (chamado doravante de perfil IIOP de grupo).

Dada a necessidade da realização destes três tipos de invocações, a IOR de grupo não pode ser estruturada da mesma forma que a utilizada nas referências a objetos convencionais (que utiliza apenas um perfil IIOP). Desta forma, um formato de IOR de grupo foi definido pela OMG como parte

das especificações. Esta nova IOR contém o perfil UIPMC e opcionalmente outros perfis IIOP. A figura 4.10 apresenta o formato de uma IOR de grupo em sua estrutura completa, com os perfis de *gateway* e de objeto que responde as requisições com resposta (perfil IIOP de grupo). A parte da IOR que identifica o grupo é representada pela estrutura `TagGroupComponent`. A IDL que define esta estrutura é apresentada na figura 4.11.

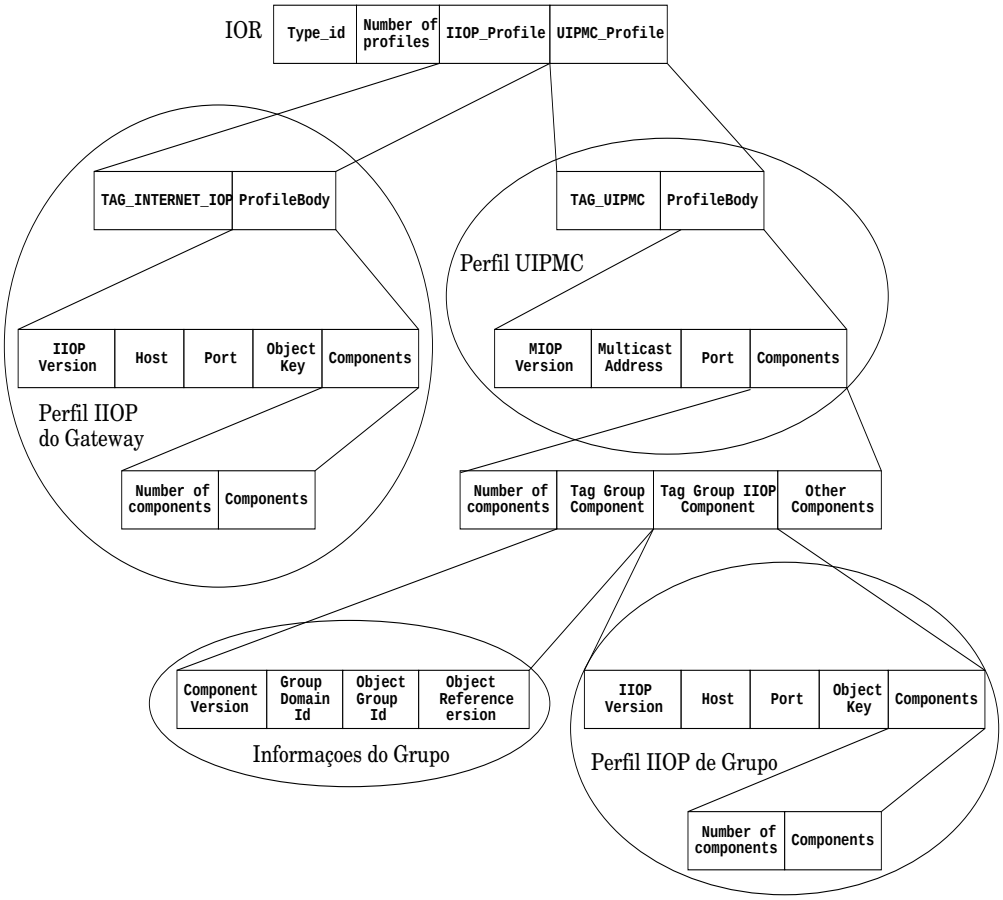

Figura 4.10: IOR de grupo *multicast*.

A descrição dos campos da estrutura `TagGroupComponent` é apresentada a seguir:

- `component_version`: A versão deste componente;
- `group_domain`: Este campo contém uma *string* que define o escopo para o identificador de grupo;
- `object_group_id`: Um identificador único de 64 bits para o grupo;
- `object_group_ref_version`: Este campo é opcional e deve conter o valor 0 quando não utilizado. Uma implementação pode utilizar este campo se múltiplas versões de uma referência a determinado grupo existirem.

```
module PortableGroup
{
  typedef GIOP::Version Version;

  struct TagGroupComponent //tag = TAG_GROUP
  {
    Version component_version;
    GroupDomainId group_domain;
    ObjectGroupId object_group_id;
    ObjectGroupRefVersion object_group_ref_version;
  };
};
```

Figura 4.11: IDL para o `TagGroupComponent`.

A criação da IOR de grupo acontece através da especificação das informações sobre o grupo (para o preenchimento das estruturas `UIPMC_ProfileBody` e `TagGroupComponent`) e das IORs com o perfil IIOP de grupo e do *gateway*. Esta criação acontece através da chamada de métodos de criação de referências do ORB (ver mais a frente o mecanismo de *corbaloc*) ou através de métodos do MGM (ver seção a frente).

4.3.2.3 Grupos de Objetos Não-Confiáveis

Um grupo de objetos representa uma coleção de implementações que compartilham um conjunto de informações em comum, associadas à definição do grupo. Normalmente estas informações são armazenadas em um gerenciador de grupos, assim, em um sistema de suporte a grupos, existem informações relacionadas ao *membership* de objetos entre as informações do grupo. Entretanto, a referência de grupo definida pela OMG nas especificações UMIOP não trata este tipo de dado (diferentemente da IOGR do FT-CORBA), e assume que estas informações serão tratadas em futuras submissões relacionadas a grupos.

4.3.2.4 Adaptador Portável de Grupo (PGA)

A fim de que o POA (adaptador de objetos padrão do CORBA) suporte a existência de grupos de objetos, quatro novas operações foram adicionadas a ele, dando origem então ao Adaptador Portável de Grupo (PGA).

Conforme já discutido, o perfil UIPMC não define uma chave de objetos única para grupos, assim o POA deve utilizar as informações associadas à estrutura `PortableGroup::TagGroupComponent`,

contida no perfil UIPMC e os identificadores de objetos locais - `PortableServer::ObjectId`'s[2] - associados as implementações dos objetos participantes deste grupo para entregar a requisição corretamente. Assim, pode-se dizer que a composição da estrutura `PortableGroup::TagGroupComponent` com os identificadores de objetos dos membros é equivalente a uma chave de objeto (`ObjectKey`) no contexto de grupos.

A figura 4.12 apresenta uma possível implementação da entrega de uma requisição a um grupo em um ORB que implementa o UMIOP.

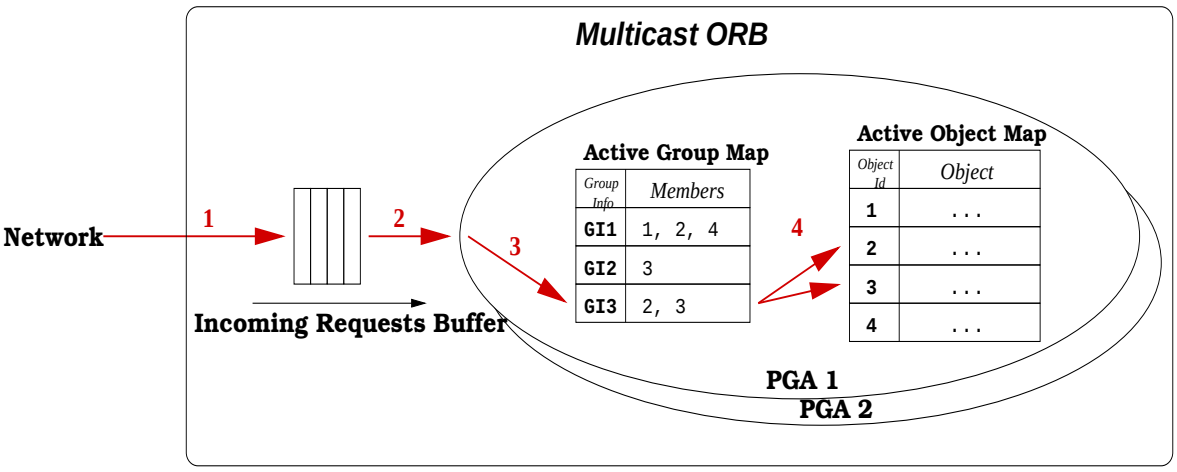

Figura 4.12: Entrega de mensagens a grupos em um ORB com MIOP.

Na figura apresentada, são definidas as várias etapas de processamento desde a chegada de uma mensagem pela rede até a entrega desta aos objetos pertencentes ao grupo. Estes vários passos são descritos a seguir:

1. Inicialmente a mensagem difundida vem do módulo de rede para a porta em que o ORB recebe requisições. Logo após a integração da coleção de pacotes MIOP que dará origem à requisição em uma mensagem GIOP, esta é colocada em uma fila de requisições;

2. Quando chega a vez da mensagem na fila de requisições, ela é retirada e passada aos vários POAs ativos no ORB[3];

3. Uma vez dentro do PGA, a estrutura `TagGroupComponent` presente no perfil UIPMC é consultada e utilizada como chave para uma busca na tabela de grupos ativos. Esta tabela mapeia `TagGroupComponent`'s para implementações. Em nosso exemplo, a requisição tem como destino o grupo cujo o `TagGroupComponent` é o `GI3`;

[2] Identificadores utilizados pelo POA e pelas aplicações para manipular objetos dentro de seu contexto[21].

[3] Aqui está uma das diferenças para com as mensagens ponto-a-ponto, pois dentro do `ObjectKey` presente nestas existe uma identificação de qual POA está gerenciando o objeto destino.

4. Pela tabela de grupos ativos, o `GI3` têm como membros os objetos 2 e 3. Nesta fase do processamento da requisição ela é repassada para estes objetos através da tabela de objetos ativos, que mapeia `ObjectId`'s para as implementações.

Na figura 4.12 vemos que a implementação da PGA necessita de uma tabela adicional em relação aos POA convencionais: a Tabela de Grupos Ativos. Esta tabela deve conter os grupos ativos neste adaptador, bem como os identificadores locais dos objetos membros deste (uma espécie de *membership* no contexto do POA).

A figura 4.13 apresenta as novas operações adicionadas ao POA para o suporte a grupos, tornando-a desta forma um PGA.

```
module PortableServer
{
  exception NotAGroupObject{};
  typedef sequence<ObjectId> IDs;

  interface POA
  {
    ...
    ObjectId create_id_for_reference(in CORBA::Object the_ref)
      raises (NotAGroupObject);
    IDs reference_to_ids(in CORBA::Object the_ref)
      raises (NotAGroupObject);
    void associate_reference_with_id(in CORBA::Object ref,
      in ObjectId oid) raises (NotAGroupObject);
    void disassociate_reference_with_id(in CORBA::Object ref,
      in ObjectId oid) raises (NotAGroupObject);
  };
};
```

Figura 4.13: Novas operações do POA.

As operações definidas na figura 4.13 permitem ao POA gerar, listar, associar e desassociar `PortableServer::ObjectId` à grupos de objetos. É importante ressaltar que todas as referências passadas ao POA como parâmetros destas novas operações devem ser referências de grupo, caso contrário a excessão `NotAGroupObject` é ativada.

4.3.2.5 MGM - Gerenciador de Grupos *Multicast*

O MGM (*Multicast Group Manager*) é o objeto de serviço opcional presente nas especificações UMIOP que define métodos de mais alto nível para o gerenciamento de grupos *multicast* e os recursos

de transporte associados a estes. Basicamente este serviço provê operações para criação e destruição de grupos e gerenciamento de propriedades.

O MGM é definido a partir de um novo módulo introduzido nas especificações UMIOP. Este módulo, chamado `PortableGroup` [22], define, de maneira uniforme, uma série de tipos, estruturas e interfaces utilizadas no tratamento de grupos dentro do CORBA.

A interface do MGM, chamada `ObjectGroupFactory`, estende três outras interfaces pertencentes ao módulo `PortableGroup`, conforme apresentado na figura 4.14.

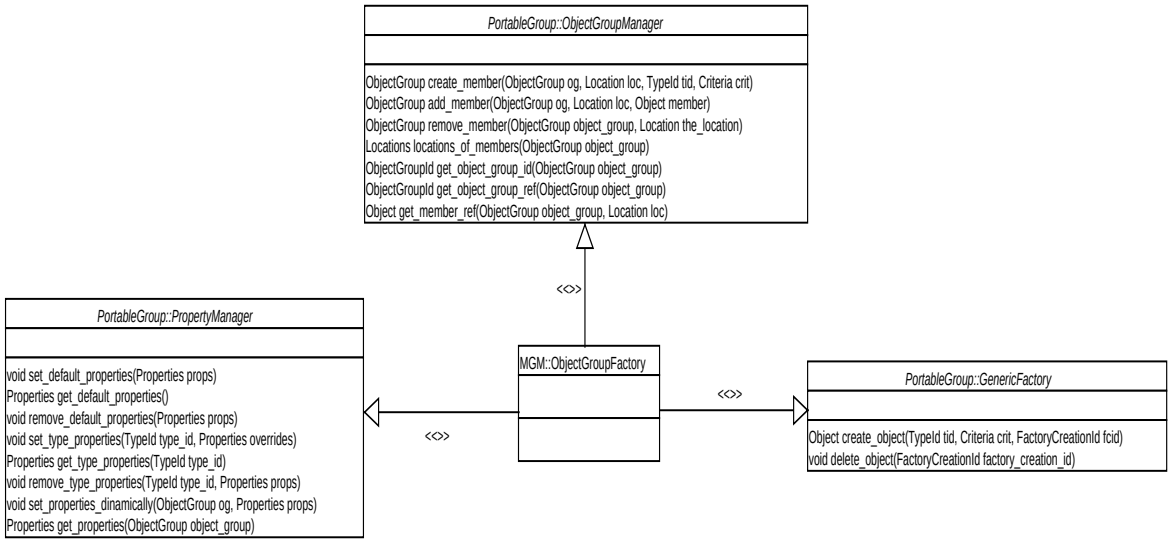

Figura 4.14: Diagrama de classes do MGM.

Na figura 4.14, a interface `ObjectGroupFactory` é basicamente a união, através da extensão, de três outras: `PropertyManager`, `ObjectGroupManager` e `GenericFactory`. Estas interfaces são descritas a seguir.

- `PortableGroup::PropertyManager`: Esta interface define métodos para o gerenciamento de três níveis hierárquicos de propriedades para objetos: propriedades padrão, propriedades do tipo e propriedades do objeto. As propriedades definidas em nível de objeto sobrepoêm as definidas para o tipo que sobrepõem as propriedades padrão. Esta interface provê vários métodos para a definição, remoção e listagem dos três níveis de propriedades;

- `PortableGroup::ObjectGroupManager`: Esta interface define métodos para o gerenciamento de grupos em geral. Entre estes métodos destacam-se os de criação, adição, remoção e obtenção de membros. Grande parte dos métodos definidos nesta interface não são utilizados no MGM (lançam a excessão `CORBA::NO_IMPLEMENT`) devido ao fato dos grupos definidos nesta especificação não manterem informações sobre *membership*. Assim, os únicos métodos utilizados

são o `get_object_group_id`, para a obtenção do identifi cador de grupo associado a referência passada, e o `get_object_group_ref`, para atualização de uma referência [4];

- `PortableGroup::GenericFactory`: Esta interface defi ne dois métodos utilizados para a criação e destruição de objetos. As propriedades iniciais do grupo são passadas ao método `create_object` através do parâmetro `Criteria`. Este parâmetro pode defi nir uma série de propriedades como o endereço *multicast* IP do grupo.

A tabela 4.1 apresenta as propriedades válidas utilizadas pelo MGM.

Propriedade	Descrição
GroupCreationMode	O endereço do grupo será criado ou defi nido?
CreateSpecifyGateway	O *gateway multicast*
SupportImplicitOperations	Objeto IIOP para operações implícitas
CreateIncludeGateway	Defi ne se o *gateway* será usado ou não
ProtocolEndpointsIPPort	Porta destino
ProtocolEndpointsIPAddress	Endereço do grupo
GroupDomainId	Escopo do grupo

Tabela 4.1: Propriedades manipuladas pelo MGM.

Vale lembrar que mesmo sendo um objeto de serviço bastante interessante para a criação e gerenciamento de grupos, o MGM é parte opcional do padrão MIOP. Os ORBs que não implementam o MGM utilizam-se do mecanismo de URLs *corbaloc* para defi nição das informações do grupo em uma *string* que é passada ao método `string_to_object` do ORB para a criação da referência ao grupo. A fi gura 4.15 apresenta um exemplo de URL válida para a criação de um grupo.

```
corbaloc:miop:1.0@1.0-MeuDominio-1/225.1.2.8:2578
```

Figura 4.15: URL que cria um grupo UMIOP.

A especifi cação completa do mecanismo de `corbaloc` para grupos UMIOP pode ser encontrada nas especifi cações [22].

4.4 Considerações Gerais sobre o FT-CORBA e o UMIOP

O FT-CORBA introduz no CORBA uma série de objetos de serviço úteis para o gerenciamento de grupos e também uma referência padronizada para grupos onde as informações de *membership*

[4] Utilizando este método, qualquer alteração de endereços ou novas alocações tornam-se válidas.

estão disponíveis. Entretanto, este padrão não aborda aspectos de comunicação de grupo, que são especialmente úteis na implementação de modelos de replicação ativa (que também não é especificada no FT-CORBA).

Como um primeiro passo para a construção de um suporte de comunicação de grupo no CORBA o padrão UMIOP foi introduzido na arquitetura. Este padrão apresenta um protocolo de difusão não confiável baseado em *multicast* IP e um modelo de grupo sem *membership*.

Em termos de modelo de grupo, é possível verificar que as referências de grupo definidas pelas duas especificações apresentam semelhanças apenas no que diz respeito à presença do componente com as informações do grupo (`TagGroupComponent` - idêntico nas duas), de resto, a presença do *membership* no IOGR do FT-CORBA e do perfil UIPMC na referência de grupo do UMIOP diferencia, de maneira cabal, estes dois tipos de referências.

Em termos de objetos de serviço, o MGM - único objeto de serviço definido pelo UMIOP (opcional na especificação) - apresenta as mesmas interfaces do Gerenciador de Replicação (SGR) do FT-CORBA. Entretanto, as operações que tem algum tipo de relação com *membership* não são implementadas no MGM. Assim, apesar de iguais em suas interfaces, o SGR apresenta um nível de complexidade e funcionalidade muitas vezes superior ao MGM.

Analisando as similaridades e diferenças entre estas duas especificações chega-se a conclusão de que, a principio, elas têm finalidades completamente diferentes: enquanto o FT-CORBA utiliza a abstração de grupos de objetos para prover serviços replicados tolerantes a faltas, o UMIOP apresenta os grupos como um conjunto de zero ou mais objetos que podem ser invocados através de difusão não confiável. Assim, o UMIOP é indicado para aplicações que fazem uso de grupos explicitamente, como aplicações de difusão de mídia, e na implementação de funcionalidades que são baseadas em *multicast* IP normalmente, como por exemplo, a descoberta de serviços em uma rede.

Mesmo com as diferenças expostas, ainda é razoável pensar no mecanismo de comunicação definido pelo UMIOP como uma primeira tentativa de se chegar a um suporte de comunicação de grupo padronizado para o FT-CORBA. A seguir são apresentados os dois principais empecílios para esta utilização:

- O MIOP, protocolo definido nas especificações UMIOP, é não confiável, portanto inadequado para utilização em serviços de replicação para tolerância a faltas;
- O modelo de objetos do UMIOP define que o protocolo MIOP só é utilizado em mensagens sem resposta (*oneway*). As mensagens com resposta devem ser enviadas para o objeto referenciado pelo perfil IIOP de grupo.

Apesar do modelo de comunicação sem respostas ser bastante natural, já que estamos falando de comunicação multiponto e simétrica, ele não é adequado ao FT-CORBA, pois a invocação de métodos com resposta é prevista para os grupos de objetos replicados nesta especificação.

A primeira limitação apresentada é decorrente das decisões de projeto do MIOP, e não pode ser vencida a não ser com extensões (ver próximo capítulo). A segunda limitação, que diz respeito às requisições com resposta, pode ser vencida através da utilização de interceptadores portáveis [21] no cliente e nos membros do grupo. Estes interceptadores desviariam mensagens com resposta enviadas à grupos UMIOP, que normalmente seriam enviadas para o objeto referenciado pelo perfil IIOP de grupo, e as difundiriam normalmente via MIOP. Nos *hosts* destino, estas requisições seriam passadas aos objetos membros do grupo normalmente, como qualquer requisição UMIOP. Entretanto, as respostas fornecidas por estes objetos seriam capturadas pelos interceptadores dos membros e enviadas a um objeto coletor (que é um objeto CORBA), instanciado no cliente, que as repassaria ao interceptador cliente, que então, as retornaria ao objeto chamador. A figura 4.16 ilustra este esquema.

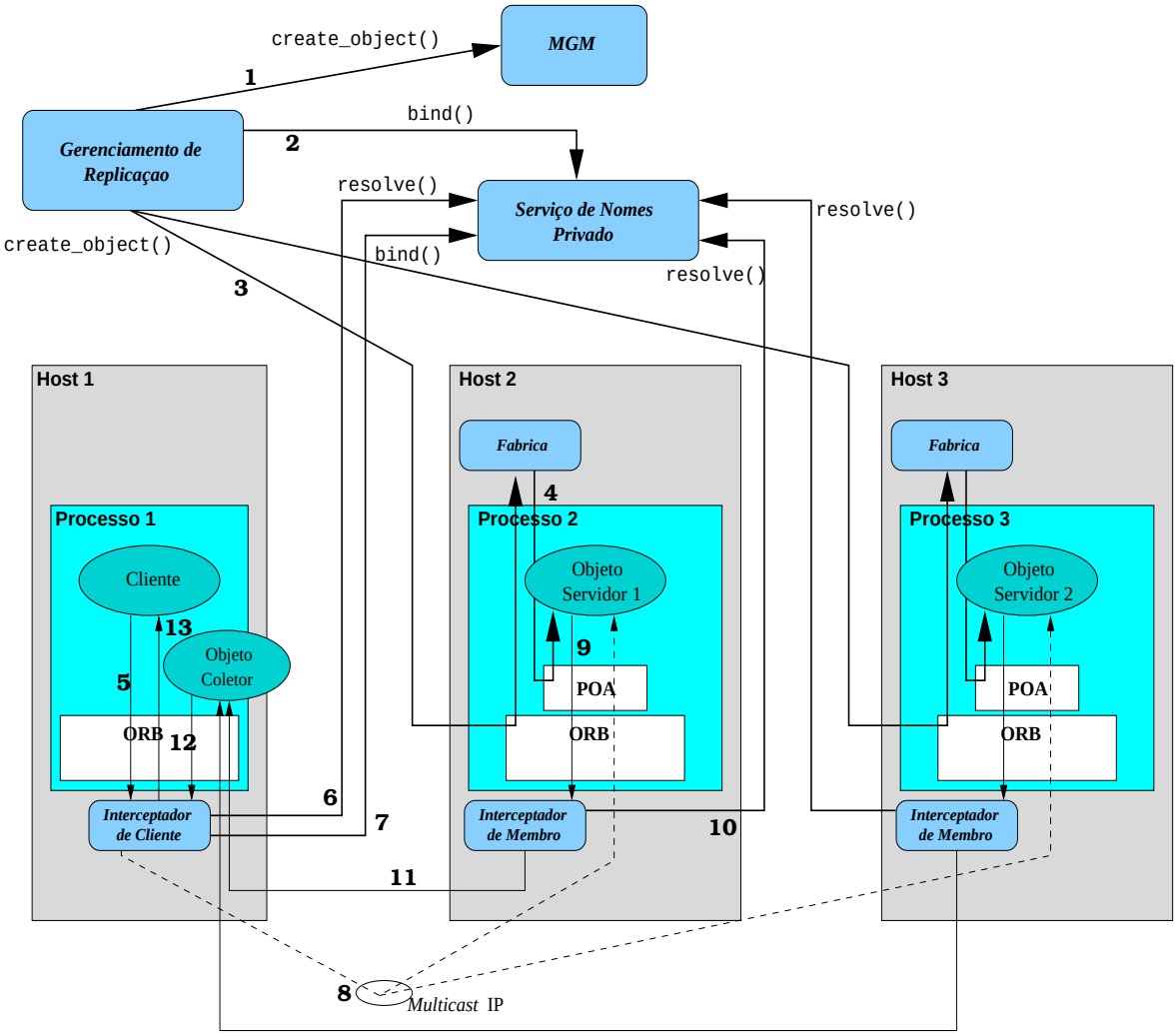

Figura 4.16: Modelo de integração do FT-CORBA com o UMIOP.

A seqüência de passos necessários para a concretização de uma requisição ao grupo através deste modelo são descritos a seguir:

1. Antes de se criar qualquer membro do grupo, o Gerenciador de Replicação, logo após a criação do grupo, deve criar um grupo UMIOP que refletirá o grupo FT-CORBA em um nível de suporte a comunicação de grupo;

2. Este grupo UMIOP é colocado a disposição em um serviço de nomes privado, só acessado pela infra-estrutura, sendo seu nome definido a partir das informações contidas na estrutura `TagGroupComponent` do IOGR;

3. Na criação dos membros, será passado a referência ao grupo UMIOP criado para que estes sejam incluídos como membros deste;

4. Quando a fábrica local cria o objeto replicado, ela o associa ao grupo UMIOP através das novas operações definidas no POA;

5. A requisição acontece via IOGR do grupo FT-CORBA obtida de alguma forma pelo cliente. Esta requisição é interceptada pelo interceptador cliente;

6. A partir das informações contidas na IOGR o interceptador cliente busca no serviço de nomes privado a referência ao grupo UMIOP correspondente ao grupo FT-CORBA invocado;

7. Encontrada a referência o interceptador verifica se a mensagem necessita de resposta. Se sim, uma referência ao objeto coletor (responsável pela recepção das respostas) deste cliente é disponibilizada no servidor de nomes privado;

8. A requisição é feita ao grupo UMIOP via *multicast* IP desconsiderando o fato dela necessitar de resposta. Esta requisição é enviada aos objetos membros em cada *host*;

9. Os objetos membros executam a requisição e enviam suas respostas de volta ao ORB para que elas sejam retornadas ao invocador;

10. O interceptador do membro intercepta este envio de resposta (que não poderia ser concretizado) e realiza uma busca no serviço de nomes pelo objeto coletor correspondente ao ORB invocador;

11. Uma vez encontrada a referência ao objeto coletor do invocador a resposta é passada a este através de uma invocação ponto-a-ponto;

12. O objeto coletor passa a resposta mais rápida ao interceptador cliente[5];

13. O interceptador cliente retransmite a resposta à operação invocada ao cliente.

Note que nesse esquema, representado na figura 4.16, o serviço de nomes privado tem papel fundamental para que o interceptador cliente possa encontrar o grupo UMIOP referente à IOGR e os

[5]Se considerarmos falhas arbitrárias, é possível se implementar um mecanismo de votação neste objeto a fim de comparar as respostas e retornar a que foi enviada pela maioria dos objetos membros.

interceptadores dos membros possam encontrar a referência ao objeto coletor, e assim despachar as respostas de volta ao cliente via IIOP. Vale lembrar também que as consultas ao serviço de nomes só são realizadas na primeira vez que este cenário é executado, já que é previsto que tanto o interceptador cliente quanto os membros utilizam mecanismos de *cache* para armazenar as referências ao grupo UMIOP e ao objeto coletor.

Para garantir a integridade do suporte de comunicação de grupo definido no esquema da figura 4.16, é interessante que existam garantias de que o serviço de nomes privado usado para armazenar as referências aos grupos UMIOP e aos coletores não esteja disponível para as aplicações, evitando, desta forma que estas referências possam ser obtidas e/ou manipuladas maliciosamente.

4.5 Conclusão

Neste capítulo foi apresentada a arquitetura CORBA e as duas principais iniciativas de se introduzir grupos nesta arquitetura. Os dois modelos de grupos definidos, respectivamente, pelas especificações do FT-CORBA e do UMIOP são bastante diferentes, principalmente no que diz respeito à semântica de confiabilidade dos grupos definidos nestas especificações, onde o UMIOP define grupos abertos, sem *membership* e com comunicação não confiável.

Foi apresentado também neste capítulo uma proposta de modelo de integração entre estas duas plataformas, entretanto, para que este modelo possa oferecer replicação ativa de forma confiável, são necessárias algumas extensões ao protocolo MIOP. O próximo capítulo apresenta os esforços deste trabalho para a criação de uma extensão do MIOP, compatível com este, que ofereça confiabilidade nas difusões.

Capítulo 5

ReMIOP - *Reliable* MIOP

Com o intuito de adequar o padrão UMIOP para sua utilização como mecanismo de comunicação de grupo do FT-CORBA foi desenvolvido um protocolo de comunicação confiável iniciado pelo receptor nos moldes do SRM e do LRMP.

Este protocolo, chamado ReMIOP, define dois tipos de mensagens de controle e um mecanismo de controle de fluxo para agregar aspectos de confiabilidade ao protocolo de difusão não confiável da OMG. Desta forma dando os primeiros passos em direção da utilização do UMIOP como suporte de comunicação de grupo do FT-CORBA.

Este capítulo trata basicamente da proposta deste protocolo e da especificação de seus algoritmos. Inicialmente serão apresentadas algumas considerações sobre o protocolo definido, em seguida uma descrição deste e de seu algoritmo de controle de fluxo é apresentada. Na parte final do capítulo temos os algoritmos utilizados pelo protocolo e o modelo de integração dele com o MIOP.

5.1 Considerações sobre o Espaço de Projeto

Considerando o espaço de projeto para serviços de difusão confiável em sistemas de larga escala [24], foram definidos alguns objetivos e características desejáveis a serem alcançados com o ReMIOP:

1. O ReMIOP considera grupos de ORBs. Desta forma, quando falamos em emissores e receptores estamos considerando ORBs emissores e ORBs receptores;

2. Não existem informações sobre os membros do grupo (*membership*). O emissor não sabe quem são os receptores e os receptores não sabem nada a respeito dos emissores. Este modelo é exatamente o mesmo do *multicast* IP e do UMIOP. Esta compatibilidade com o padrão UMIOP permite a utilização da mesma referência de grupo inalterada pelo ReMIOP;

3. Não deve haver confirmação de recepção de mensagens. Se o emissor não receber pedidos de retransmissão ele considera que os pacotes MIOP estão sendo entregues sem nenhum problema. A utilização de confirmação não é possível pois ela só seria eficaz se o *membership* estivesse disponível, o que conforme apresentado no item anterior, não é o caso;

4. Inicialmente o protocolo está sendo desenvolvido para utilização em replicação ativa [43] no contexto de tolerância a faltas. Assim sendo o número de membros de um grupo não deve passar de algumas dezenas, logo a escalabilidade não é um requisito fundamental no projeto do ReMIOP, apesar de ela ser altamente desejável;

5. O protocolo deve garantir confiabilidade e desempenho tanto em redes locais como na Internet. Mesmo considerando que teremos poucos membros no grupo, eles podem estar espalhados por regiões distantes e conectados via a Internet pública;

6. Apesar de confiável, o ReMIOP não faz nenhuma garantia a respeito da ordenação das mensagens (propriedades de ordenação). Garantias de ordem devem ser objeto de futuras extensões do ReMIOP;

7. O ReMIOP não pode prover qualquer garantia a respeito do tempo de entrega dos pacotes (propriedades de temporização). Esta decisão reflete o fato da Internet não definir mecanismos de garantia de tempo, a não ser via reserva de recursos (que não estão sempre disponíveis), que não são considerados pelo protocolo;

8. O ReMIOP funciona acoplado ao MIOP, que só pode ser utilizado em *hosts* que implementam o *multicast* IP;

9. O ReMIOP é um protocolo iniciado pelo receptor, e se utiliza de NACKs enviados aos emissores para sinalizar retransmissões. Além disso, mecanismos de supressão de NACKs e retransmissões locais (fazendo com que receptores também possam atender a requisições de NACKs) são usados para diminuir o número de NACKs e retransmissões no grupo;

10. O ReMIOP não mantém os membros dos grupos divididos hierarquicamente, ou em qualquer outro tipo de organização;

11. O controle de congestionamento e da taxa de transmissão é realizado pelo emissor baseando-se no número de pacotes perdidos pelos membros do grupo, em especial os mais lentos (aqueles que perderam mais pacotes);

12. Não é feita nenhuma consideração sobre segurança. Este aspecto é deixado, em um nível mais elevado, para o ORB ou as aplicações e, em um nível mais baixo, para o IGMP (*Internet Group Management Protocol*) e os controles que ele provê para inclusão de membros nos grupos *multicast* IP.

Estas considerações visam limitar o escopo do protocolo seguindo as recomendações do IETF [33] que não mais esta trabalhando em um protocolo de transporte confiável multiponto e sim em um conjunto de componentes que devem ser agrupados convenientemente para a construção de protocolos com características desejáveis para o tipo de aplicação em questão [50].

5.2 O Protocolo

O ReMIOP visa dar confiabilidade ao MIOP acrescentando duas funcionalidades à este:

- **Retransmissões:** Como a possibilidade de perda de mensagens existe e é considerável em sistemas de larga escala o ReMIOP inclui um tipo de mensagem de controle para o pedido de retransmissões. Esta mensagem contém o identificador dos pacotes MIOP extraviados, assim, quem a receber, se tiver os pacotes pedidos, pode difundi-los novamente no grupo;
- **Controle de Fluxo:** Um mecanismo de controle de fluxo simples foi incluído com o intuito de evitar sobrecargas e minimizar a perda de pacotes em regiões congestionadas da rede. Através desse algoritmo de controle do fluxo e da premissa (informal) de que o sistema opera a maior parte do tempo em condições normais (sem congestionamentos e falhas de omissão de envio), é possível garantir que todas as mensagens enviadas chegarão aos membros do grupo.

O protocolo opera da mesma maneira que os protocolos iniciados pelo receptor descritos no capítulo 3. As mensagens (pacotes MIOP) são difundidas pelo emissor no grupo sem saber quais são seus membros. Os receptores detectam perdas de mensagens quando encontram lacunas na seqüência das mensagens recebidas. Quando é detectada a falta de uma mensagem, uma mensagem de controle (NACK) é difundida no grupo pedindo a retransmissão. Quem receber esta mensagem, seja o emissor ou algum receptor, e tiver a mensagem requerida, a difunde novamente no grupo, para que os que a perderam tenham possibilidade de recebê-la. Este protocolo inclui ainda mensagens de controle difundidas periodicamente no grupo pelos receptores para reportar o estado de seus *buffers* aos emissores, a fim de que estes possam ajustar sua taxa de envio através do algoritmo de controle de fluxo.

Além do mecanismo básico de um protocolo iniciado pelo receptor, algumas otimizações foram adicionadas ao protocolo a fim de conseguir um melhor desempenho. Dentre estas modificações podemos citar a utilização do RINA, que evita explosões de NACKs, e a postergação das retransmissões por um tempo aleatório, diminuindo o número de retransmissões para uma mensagem perdida que tenha sido difundida no grupo.

5.3 As mensagens de Controle ReMIOP

O ReMIOP foi definido na forma de vários algoritmos, um para cada função a ser realizada no protocolo. Foi definida também uma estrutura de dados `ReMIOPData` para as mensagens de controle do ReMIOP. A figura 5.1 apresenta a IDL que define esta estrutura.

```
module ReMIOP {

    enum MessageType{
        NACK,
        STATE
    };

    struct MIOPMessage{
        string senderId;
        unsigned long long sequenceNumber;
    };

    typedef sequence<MIOPMessage> Messages;

    struct ReMIOPData {
        MessageType type;
        string senderId;
        Messages msgs;
    };
};
```

Figura 5.1: IDL com a definição das mensagens de controle do ReMIOP.

Os dois tipos de mensagens de controle citados acima são:

- **NACK:** É uma mensagem de controle utilizada para pedir retransmissões de mensagens. Mensagens deste tipo tem no campo `msgs` da estrutura `ReMIOPData` o identificador do pacote MIOP que este faltando. Um NACK é escalonado para envio por algum receptor quando é detectada uma falha na seqüência de mensagens.
- **STATE:** Mensagens de estado são enviadas periodicamente por cada receptor para informar seu estado aos emissores do grupo. As mensagens de estado são importantes para que todos os participantes do grupo possam detectar quais mensagens são estáveis e assim liberar espaço nos `buffers`. O campo `msgs` da estrutura `ReMIOPData` contém os maiores identificadores das mensagens estáveis[1] referentes à cada emissor conhecido pelo receptor;

[1] **Mensagem estável**, neste contexto, é o nome dado a uma mensagem tal que não existam mensagens não recebidas com número de seqüência menor que o dela em um dado *buffer*.

Estes dois tipos de mensagens de controle são encapsuladas em pacotes MIOP, da mesma forma que o fragmento da mensagem GIOP é encapsulado nas mensagens de dados, e difundidos no grupo.

5.4 Controle de Fluxo

O controle de fluxo é um ponto fundamental em qualquer mecanismo de difusão confiável. Através de mecanismos de controle de fluxo é possível evitar congestionamentos e a explosão de NACKs que isso pode acarretar na rede. Além disso, um controle de fluxo bem feito pode evitar problemas com enchimento de *buffers* e descartes de pacotes.

O mecanismo empregado para realizar o controle de fluxo no ReMIOP utiliza mensagens de estado, que contém o estado dos *buffers* de recepção dos membros dos grupos[2]. Através destas informações o emissor recebe dados a respeito da "velocidade"de seus receptores, e usa uma função de atualização da taxa de transmissão de acordo com a capacidade dos receptores. Este mecanismo se utiliza de dois tipos de *buffers*: um para emissores e outro para receptores.

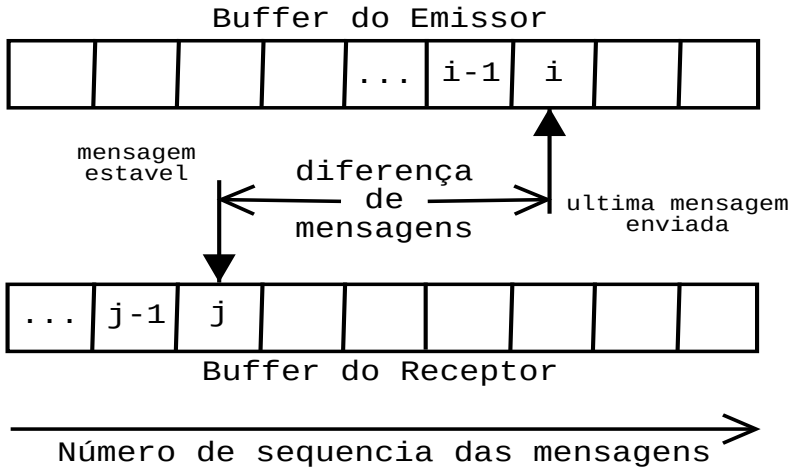

Figura 5.2: O controle de fluxo no ReMIOP.

Na figura 5.2 o *buffer* do emissor, também chamado *buffer* de envio, com tamanho fixo, é utilizado para armazenar as mensagens enviadas e por enviar. O tamanho deste *buffer* define quantas mensagens antigas podem ser reenviadas por ele em caso de pedidos de retransmissão. No receptor, o *buffer* serve para armazenar as mensagens recebidas. A diferença entre o número de seqüência da última mensagem enviada pelo emissor (i no *buffer* de envio), e a maior mensagem estável (j no *buffer* de recepção) é o parâmetro utilizado para ajustar a taxa de envio das mensagens.

Seja a diferença $\delta = i - j$ de tal modo que: quanto maior δ, menor deve ser a taxa de envio para que os receptores lentos (cujo j é muito menor que i) consigam "consumir"corretamente as mensagens do

[2]Cada receptor tem um *buffer* de recepção para cada emissor.

emissor. O principal objetivo deste algoritmo é evitar retransmissões ajustando a taxa de envio de tal forma que todos os membros do grupo, mesmo os que estão em áreas congestionadas da rede possam receber as mensagens.

O algoritmo de controle de fluxo do ReMIOP é inspirado no mecanismo definido para o LRMP [29]. A taxa de transmissão dos emissores varia sempre entre os valores $[R_{min}, R_{max}]$. Onde R_{min} e R_{max} são definidos pela aplicação. A taxa de transmissão inicial é definida como $R = (R_{min} + R_{max})/2$, e a cada 8 envios de pacotes R é incrementado em $0.125R$. A cada recepção de uma mensagem de estado, o emissor ajusta a taxa de transmissão (R) de acordo com a seguinte regra (onde *size* é o tamanho do *buffer* do emissor, i.e. sua capacidade máxima):

$$new\_rate(R, \delta, size) = \begin{cases} R & \text{se } \delta \leq size/5 \\ 0.75R & \text{se } size/5 < \delta \leq size/4 \\ 0.50R & \text{se } size/4 < \delta \leq size/3 \\ 0.25R & \text{se } size/3 < \delta \end{cases} \quad (5.1)$$

Na equação 5.1, a função *new_rate* define uma nova taxa de envio baseando-se na taxa atual (R), no δ e no tamanho do *buffer* (*size*).

A regra de ajuste de fluxo apresentada mostra que a taxa de envio em um emissor é sempre definida pelo receptor mais lento (aquele para o qual δ é maior). Desta forma o protocolo é vulnerável a receptores maliciosos que podem causar ataques do tipo DoS (*Denial of Service*) na medida em que eles exigem taxas de envios de dados cada vez mais lentas no emissor. Inicialmente a única solução encontrada para tolerar estas intrusões é estabelecer que, independentemente da diferença nas janelas (δ), o emissor nunca utilizará uma taxa de transmissão menor que R_{min}.

5.5 Especificação dos Algoritmos

Esta seção apresenta a especificação dos algoritmos utilizados na implementação do ReMIOP.

Os algoritmos do ReMIOP trabalham com grupos de processos, um grupo $G = \{p_1, p_2, ..., p_n\}$ é um conjunto de processos contido em $\mathcal{P}$, que representa o conjunto de todos os processos do sistema. Cada grupo tem um identificador denotado por $id(G)$. Este identificador é utilizado nas primitivas de difusão não confiável.

Toda especificação apresentada nesta seção considera que existem três tipos de mensagens manipuladas pelo ReMIOP: mensagens de dados, que são representadas pelo conjunto $\mathcal{D}$, pedidos de retransmissão (ou NACKs), representadas pelo conjunto $\mathcal{N}$ e mensagens de estado, representadas pelo conjunto $\mathcal{S}$. Uma mensagem m é representada nos algoritmos pela tupla $< sender, seq, size, cids >$, onde *sender* é o processo emissor da mensagem ($m.sender \in \mathcal{P}$), *seq* é o número de seqüência local da

mensagem (um número natural), *size* é o tamanho da área de dados da mensagem em bytes (também um número natural) e *cids* é um conjunto de identificadores de mensagens[3], utilizado nas mensagens de controle ($m \in \mathcal{N} \cup \mathcal{S}$) e tendo valor $\emptyset$ nas mensagens de dados ($m \in \mathcal{D}$). O identificador de uma mensagem m, denotado por $m.id$, é dado pela tupla $< sender, seq >$ tal que $m.sender = m.id.sender$ e $m.seq = m.id.seq$.

Antes da apresentação dos algoritmos do ReMIOP, faz-se necessária uma breve descrição das variáveis globais utilizadas no protocolo:

- As taxas de transmissão mínima e máxima são representadas pelas variáveis R_{min} e R_{max} respectivamente. Os valores destas taxas são expressos em bytes por unidades de tempo. Estas variáveis são inicializadas com parâmetros definidos durante a inicialização do sistema;
- Existem três parametros de tempo definidos no sistema, são eles: P_{repair}, P_{nack} e P_{state}. Os dois primeiros definem os limites de tempo de espera para o envio de NACKs e mensagens restauradoras (atendendo à NACKs), já o terceiro define com que freqüência as mensagens de estado serão enviadas pelos receptores. Estes parâmetros são definidos durante a iniciação do sistema;
- A constante *size* define a capacidade dos *buffers* usados nos algoritmos em termos de número de mensagens;
- R_G é a taxa de transmissão do emissor para o grupo G. Inicialmente esta variável tem como valor $(R_{min} + R_{max})/2$, porém é alterada várias vezes durante a execução do protocolo de acordo com o algoritmo de controle de fluxo;
- A variável N_G representa o número de difusões realizadas pelo emissor no grupo G. Seu valor inicial é nulo;
- A variável $last_G$ representa o número de seqüencia da última mensagem difundida pelo emissor no grupo G. Seu valor inicial é nulo;
- T_G marca o instante de tempo local da última difusão realizada pelo emissor no grupo G. Seu valor inicial é nulo;
- O conjunto M_G^p contém as mensagens difundidas pelo processo p no grupo G. Existem vários conjuntos nos receptores, um para cada emissor conhecido por este;
- O conjunto de mensagens a serem difundidas pelo emissor que está executando o protocolo é denotado por M_{send}.

[3]Este campo corresponde ao campo `msgs` da estrutura `ReMIOPData`.

O emissor que deseja difundir uma mensagem m em um grupo G deve executar o algoritmo 3. Neste algoritmo, que deve ser executado de maneira não reentrante, são realizadas basicamente três passos:

Algoritmo 3 Difusão de uma mensagem m em um grupo G

1: {Para executar *R-multicast*(G,m)}
2: $d \leftarrow max(m.size/R_G + T_G - T(t), 0)$
3: $T_G \leftarrow T(t) + d$
4: $M_{send} \leftarrow M_{send} \cup \{m\}$
5: $schedule\_multicast(G,m,d)$

1. Cálculo do tempo estimado a ser esperado para a execução da difusão (linha 2). Este tempo de espera, representado pela variável d, serve para que o emissor respeite sua taxa de envio para o grupo. O cálculo de d é feito com base no tamanho da mensagem enviada ($m.size$), na taxa de envio R_G, e na diferença entre o instante da última difusão realizada (T_G) e o tempo atual lido do relógio local do processo ($T(t)$). A função *max*, que retorna o maior valor entre dois números, é utilizada para evitar que d receba valores negativos;

2. Adição da mensagem a ser enviada no conjunto de mensagens a serem enviadas: M_{send} (linha 4);

3. Escalonamento da difusão para ser executada após d unidades de tempo (linha 5).

Note que segundo o algoritmo 3, o valor de d será nulo, resultando na difusão imediata da mensagem, sempre que a aplicação (que chama *R-multicast*(G,m)) respeitar a taxa de envio R_G.

O algoritmo 4 apresenta os passos necessários para o escalonamento da difusão de uma mensagem.

Algoritmo 4 Escalonamento da difusão da mensagem m em um grupo G para $T(t)+d$

1: {Para executar schedule_multicast(G,m,d)}
2: $delay(d)$
3: **if** $m \in M_{send}$ **then**
4: $M_{send} \leftarrow M_{send} - \{m\}$
5: *U-multicast*$(id(G), m)$
6: $last_G \leftarrow m.seq$
7: $N_G \leftarrow N_G + 1$
8: **if** $N_G mod 8 = 0$ **then**
9: $R_G \leftarrow min(1.125R, R_{max})$
10: **end if**
11: **end if**

Neste algoritmo, o sistema aguarda o tempo especificado (d) e depois verifica se a mensagem a ser enviada ainda pertence ao conjunto M_{send}, esta verificação é necessária pois a mensagem pode ter

sido difundida no grupo por outro membro em caso de retransmissões, não sendo mais necessário seu envio. Se o envio da mensagem ainda for necessário então ela é difundida (linha 5), o seu número de seqüência é armazenado (linha 6) e o contador de difusões é incrementado (linha 7), aplicando-se a seguir a regra de que a cada 8 difusões a taxa do grupo é incrementada em $0.125R$, até um máximo de R_{max} (linhas 8 e 9).

O algoritmo para agendamento de difusões, deve ser executado em uma *thread* separada a fi m de não bloquear a aplicação durante o tempo de espera d.

O algoritmo 5 apresenta os passos executados para liberação de mensagens.

Algoritmo 5 Algoritmo de recepção e liberação de mensagens.

1: {Para executar $R\text{-}deliver(m)$}
2: $U\text{-}receive(id(G), m)$
3: $M_{send} \leftarrow M_{send} - \{m\}$
4: **if** $m \in \mathcal{D}$ **then** {Tratamento de mensagens de dados (caso 1)}
5: $M_{send} \leftarrow M_{send} - \{m_{nack} \in \mathcal{N} | m.id \in m_{nack}.cids\}$
6: $M_G^{m.sender} \leftarrow M_G^{m.sender} \cup \{m\}$
7: $R\text{-}deliver(m)$
8: $call\_repair(G, M_G^{m.sender})$
9: **else if** $m \in \mathcal{S}$ **then** {Tratamento de mensagens de estado (caso 2)}
10: $S_{this} \leftarrow \{id \in m.cids | id.sender = this\}$
11: **if** $S_{this} \neq \emptyset$ **then**
12: $j \leftarrow id_s.seq : id_s \in S_{this}$ {Maior mensagem estável}
13: $i \leftarrow last_G$ {Última mensagem enviada}
14: $R_G \leftarrow max(new\_rate(R_G, i - j, size), R_{min})$
15: **end if**
16: **else if** $m \in \mathcal{N}$ **then** {Tratamento de mensagens de nack (caso 3)}
17: **if** $\exists id_r \in m.cids \wedge \exists m_r \in M_G^{id_r.sender} (m_r.id = id_r \wedge m_r \notin M_{send})$ **then**
18: $M_{send} \leftarrow M_{send} \cup \{m_r\}$
19: $schedule\_multicast(G, m_r, random(P_{repair}, 2P_{repair}))$
20: **end if**
21: **end if**

A primeira ação a ser executada no algoritmo 5, independentemente do tipo de mensagem recebida, é a retirada desta do conjunto de mensagens a serem enviadas[4] (linha 3). Depois desse ponto o algoritmo trata três casos distintos de recepção de mensagens, enumerados a seguir:

1. Se a mensagem recebida contém dados ($m \in \mathcal{D}$), são cancelados possíveis NACKs agendados referentes à esta mensagem (linha 5), ela é armazenada no *buffer* de mensagens recebidas de seu emissor (linha 6) e então é liberada para a aplicação (linha 7). Depois disso, o sistema verifi ca se existe alguma mensagem deste emissor faltando e agenda NACKs através do procedimento *call_repair* (apresentado a seguir);

[4]Esta retirada das mensagens recebidas justifi ca a verifi cação após o tempo de espera d em *schedule_multicast*.

2. Se a mensagem recebida for uma mensagem de estado ($m \in \mathcal{S}$), que contém identificadores das maiores mensagens estáveis para cada emissor, o processamento ocorre da seguinte maneira: o maior número de seqüência de uma mensagem estável referente a este processo (representado por *this*) é obtido (se estiver disponível obviamente) e atribuído a j (linhas 10-12), o número de seqüência da última mensagem enviada é atribuido a i (linha 13), e finalmente a taxa de transmissão é atualizada de acordo com a função *new_rate*, definida pela equação 5.1. A idéia básica é obter os valores de i e j para então calcular o novo fluxo, respeitando o limite mínimo R_{min};

3. Se a mensagem recebida for um NACK, então é verificado se a mensagem requisitada está disponível e se ela não está escalonada para envio (linha 17), se esta condição for sustentada, a mensagem requerida é colocada no conjunto de mensagens a difundir (linha 18) e uma difusão é agendada para dali a $[P_{repair}, 2P_{repair}]$ unidades de tempo[5] (linha 19). Desta forma, se esta requisição for atendida por outro membro do grupo durante este período, a retransmissão da mensagem é cancelada.

A utilização de um temporizador aleatório para a difusão de mensagens de reparo (em resposta a NACKs), cujos limites são definidos pela variável P_{repair}, é importante para evitar que o grupo todo responda aos NACKs e haja uma explosão de retransmissões. Desta forma o valor de P_{repair} deve ser configurado de tal forma a balancear o tempo de espera para o envio das reparações e o tamanho do intervalo do qual este tempo será escolhido[6].

O algoritmo 6 encontra lacunas na seqüência de mensagens recebidas e envia pedidos de retransmissão (NACKs) para as mensagens faltosas.

Algoritmo 6 Detecção de lacunas em *msgset* e difusão de NACKs em G.

1: {Para executar call_repair(G,M):}
2: $m_{min} \leftarrow m \in \{m_1 \in M | \forall m_2 \in M(m_1.seq \leq m_2.seq)\}$
3: $nrecv \leftarrow \{m_1 \in \mathcal{D} - M | \forall m_2 \in M(m_{min}.seq < m_2.seq \wedge m_2.seq = m_1.seq + 1)\}$
4: **for all** $m_{nr} \in nrecv$ **do**
5: $\quad m_{nack} \leftarrow m \in \{m_1 \in \mathcal{N} \,|\, m_1.cids = \{m_{nr}.id\}\}$
6: $\quad M_{send} \leftarrow M_{send} \cup \{m_{nack}\}$
7: $\quad schedule\_multicast(G, m_{nack}, random(P_{nack}, 2P_{nack}))$
8: **end for**

As linhas 2 e 3 do algoritmo encontram as mensagens faltantes no conjunto de mensagens: na linha 2 a mensagem com número de seqüência menor (m_{min}) é encontrada e na linha 3 o conjunto de mensagens não disponíveis é definido a partir de M e m_{min}.

[5] Um valor aleatório é escolhido deste intervalo através da função *random*.

[6] Quanto maior o intervalo $[P_{repair}, 2P_{repair}]$ menor a quantidade de retransmissões de uma mesma mensagem requisitada devido aos valores diferentes dos tempos de espera escolhidos em cada membro do grupo.

Os NACKs criados pelo algoritmo 6 são agendados para envio dali a um tempo entre $[P_{nack}, 2P_{nack}]$. Este tempo aleatório é parte do mecanismo RINA utilizado. Se um NACK idêntico a este chegar (algum outro membro do grupo já requisitou a mensagem que falta) o envio da mensagem é cancelado (linha 3 do algoritmo 5).

O último algoritmo a ser apresentado é o de envio de mensagens de estado. O algoritmo 7, apresenta o procedimento de envio de mensagens de estado para o grupo. Cada receptor membro do grupo deve executar este algoritmo indefinidamente.

Algoritmo 7 Algoritmo de difusão de mensagens de estado para o grupo G.

1: {Para que executar state(G):}
2: **loop**
3: $delay(P_{state})$
4: $m_{state}.cids \leftarrow \emptyset | m_{state} \in \mathcal{S}$
5: **for all** M_G^p **do**
6: $m_{min} \leftarrow m \in \{m_1 \in M_G^p | \forall m_2 \in M_G^p (m_1.seq \leq m_2.seq)\}$
7: $stable \leftarrow \{m.id | (m \in M_G^p) \wedge (\forall m_1 \in M_G^p \exists m_2 \in M_G^p ((m_{min}.seq < m_1.seq \leq m.seq) \wedge (m_1.seq = m_2.seq + 1)))\}$
8: $m_{state}.cids \leftarrow m_{state}.cids \cup stable$
9: **end for**
10: $schedule\_multicast(G, m_{state}, 0)$
11: **end loop**

Neste algoritmo temos o procedimento de envio de mensagem de estado com as maiores mensagens estáveis[7] no receptor para cada emissor. Este procedimento é repetido indefinidamente e executado a aproximadamente cada P_{state} unidades de tempo (linha 3). O procedimento funciona da seguinte maneira: uma mensagem de estado é inicializada com seu campo *cids* como $\emptyset$ (linha 4) e então o conjunto de identificadores estáveis é construído: um identificador para cada *buffer* de emissor conhecido. Cada um destes identificadores é obtido através de três passos (executados dentro do *loop for*):

1. Dentro do *buffer* escolhido encontra-se a mensagem com o menor número de seqüência (m_{min}) (linha 6);
2. A maior mensagem estável m (obtida no conjunto de cardinalidade 1 $stable$) é aquela para a qual não existem mensagens faltando com número de seqüência maior que $seq(m_{min})$ e menor $seq(m)$ (linha 7);
3. O identificador da mensagem estável é incluído no conjunto de identificadores que serão enviados ($m_{state}.cids$)) (linha 8).

[7] Mensagens estáveis com maior identificador.

O parâmetro P_{state} é responsável direto pela freqüência com que mensagens de estado são geradas por cada membro do grupo, e deve ser ajustado convenientemente a fi m de que o sistema possa manter um bom controle de congestionamento e ao mesmo tempo não sobrecarregue o meio de comunicação com uma grande quantidade de mensagens de estado.

5.6 Integração do ReMIOP com o MIOP

As mensagens de controle ReMIOP são pacotes MIOP difundidos normalmente para o grupo *multicast* IP. Entretanto alguns cuidados básicos são tomados no preenchimento dos campos dos cabeçalhos MIOP destes pacotes para que emissores e receptores que não implementem o ReMIOP possam participar de grupos onde ele é utilizado. Estes *hosts* devem ignorar os pacotes de controle e trabalhar normalmente com as mensagens convencionais MIOP.

A seguir, são apresentados os valores a serem utilizados para preenchimento do cabeçalho MIOP (estrutura `MIOP::PacketHeader_1_0`) das mensagens de controle ReMIOP.

- O identifi cador de mensagens é sempre o mesmo e não deve ser utilizado por mensagens de dados (campo `Id`);
- O campo `packet_number` é defi nido com 0;
- O campo `number_of_packets` é defi nido com 2;
- No campo `flags` é marcado o bit mais signifi cativo para indicar que o pacote em questão é uma mensagem de controle ReMIOP.

Os demais campos do pacote MIOP são preenchidos de maneira convencional de acordo com os dados enviados e a especifi cação do protocolo.

Defi nindo os campos do cabeçalho MIOP convenientemente os receptores de pacotes ReMIOP que tiverem condição de processar estas mensagens perceberão através do campo `flags` a natureza do pacote e os tratará adequadamente. Os receptores que não implementam o ReMIOP entenderão o pacote como o primeiro elemento de uma coleção de pacotes de tamanho 2. Como o segundo pacote dessa coleção nunca vai chegar, ela nunca vai ser liberado para as camadas superiores do ORB, e será descartada por ocasião de seu *timeout*, de acordo com as especifi cações do MIOP [22].

5.7 Conclusão

Este capítulo apresentou o ReMIOP como uma extensão do MIOP especifi cado pela OMG. O ReMIOP oferece, além da funcionalidade básica de difusão seletiva defi nida pelo MIOP, serviços

de pedido de retransmissão e de controle de fluxo. Estes serviços dão uma maior confiabilidade ao sistema de comunicação.

O ReMIOP foi desenvolvido dentro da filosofia da OMG, sem alterar o MIOP e mantendo compatibilidade com este. Para se atingir este resultado final foi utilizado exatamente o mesmo modelo de grupo definido nas especificações do UMIOP: grupo aberto e sem *membership*.

O próximo capítulo apresenta uma descrição das implementações realizadas do MIOP e do ReMIOP.

Capítulo 6

Implementação e Resultados

O padrão UMIOP foi introduzido em outubro de 2001. Assim, no inicio de nossos experimentos não havia um ORB disponível que implementasse este padrão. O que nos levou a um esforço para integrar o protocolo MIOP e seu modelo de grupo em um ORB de código aberto, para posteriormente estender esta implementação a fim de dar suporte também à difusão confiável.

Este capítulo apresenta estes esforços, através da descrição de alguns pontos do projeto do ORB *multicast* MJACO. Ao final do capítulo são apresentados alguns resultados de testes com este ORB.

6.1 MJACO

O **MJACO** [2] é uma extensão do *JacORB* [6], um ORB CORBA convencional de código aberto que implementa a especificação CORBA 2.3. A arquitetura do MJACO foi definida de tal forma a permitir o convívio de duas pilhas de protocolos (IIOP/TCP/IP e MIOP/UDP/*multicast* IP) no mesmo ORB, contribuindo, deste modo, para uma melhor interoperabilidade e portabilidade.

Esta seção apresenta os principais aspectos relacionados à implementação do MJACO. Inicialmente será definido o modelo de integração adotado na implementação do UMIOP neste e depois uma breve descrição da arquitetura do *JacORB* e das modificações e adições feitas para implementação do MJACO.

6.1.1 Abordagem de Integração

A figura 6.1 apresenta a arquitetura de integração do UMIOP ao ORB definida neste trabalho e implementada no projeto MJACO. Nesta figura temos o ORB com as duas pilhas de protocolos: uma para a comunicação ponto-a-ponto, baseada em IIOP, utilizando os serviços do TCP/IP, e outra

para comunicação multiponto formada pelo MIOP, que utiliza UDP/*multicast* IP como mecanismo de transferência de seus pacotes. O nosso modelo de integração apresenta os vários elementos definidos na especificação que compõem o suporte para os dois modelos de comunicação. Foram adicionados ainda outros componentes e extensões, não definidas na especificação, cuja finalidade é facilitar o convívio das diferentes pilhas e melhorar a eficiência de utilização do conjunto.

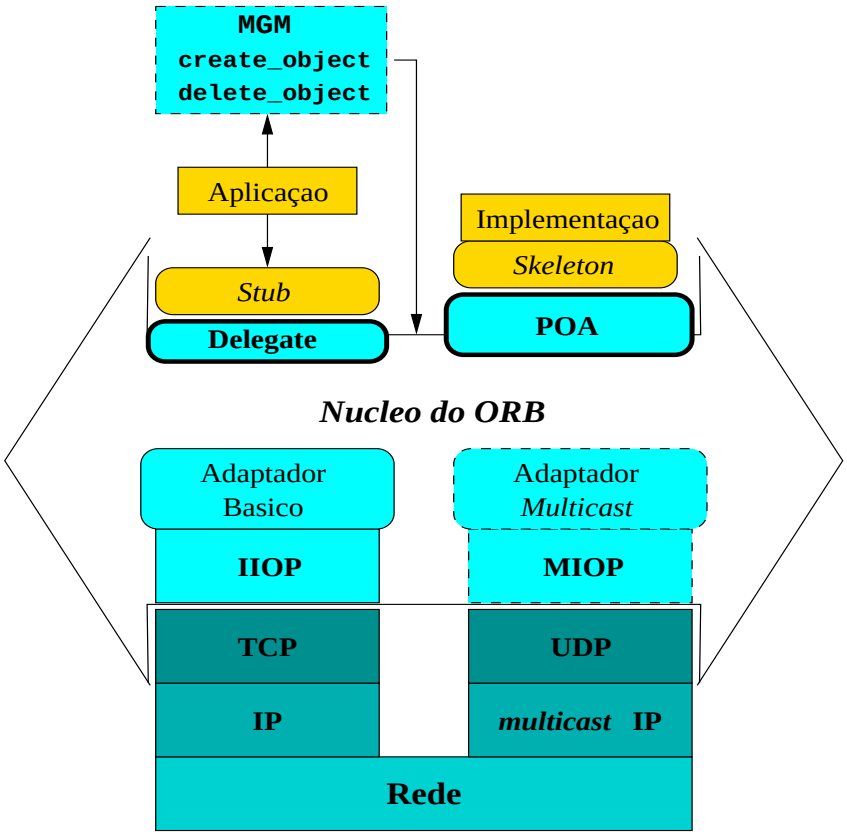

Figura 6.1: Arquitetura do MJACO.

Um componente fundamental que faz parte do nosso modelo de integração é o **Adaptador *Multicast***, responsável pela gestão dos *sockets multicast* utilizados na recepção de pacotes MIOP e pela entrega de mensagens endereçadas a grupos aos POAs ativos no ORB. O módulo UMIOP cumpre as tarefas, previstas nas propostas de padrão, no que se refere à tradução das mensagens GIOP em coleções de pacotes MIOP e vice-versa.

O POA e o *Delegate* são os principais componentes do ORB a serem alterados para a introdução do UMIOP. A alteração do *Delegate* se dá em alguns pontos para dar suporte ao envio de mensagens GIOP para grupos, já que ele é o primeiro componente interno ao ORB a ser ativado quando ocorre uma chamada de método no *stub*. Em nossa abordagem, é nele que é escolhida qual das pilhas de protocolos será utilizada para envio de uma mensagem GIOP correspondente ao método chamado. O

POA, por sua vez, além do acréscimo das quatro primitivas descritas nas especificações da OMG, deve ser alterado para o processamento de requisições endereçadas a grupos, realizando buscas na tabela de grupos ativos para obtenção das implementações membros do grupo para o qual a mensagem está endereçada. O POA também é o componente responsável pela ativação do adaptador *multicast* para a execução das operações de gerenciamento definidas pelo *multicast* IP. Por exemplo, quando a operação `associate_reference_with_id` é chamada para o registro do primeiro membro de um grupo no ORB, o POA chama o adaptador *multicast* para que este crie um *socket* e execute a operação `JoinGroup` do *multicast* IP no endereço definido no perfil UIPMC presente na referência de grupo. A partir de então, o ORB passa a receber mensagens destinadas a este grupo. As outras operações do POA relacionadas à grupos, também resultam na execução de operações de gerenciamento do *multicast* IP.

Este modelo de integração foi concretizado no *JacORB*, dando origem ao MJACO, mas pode ser reproduzido em outros ORBs com poucas modificações, já que os principais componentes do ORB alterados são padronizados e estão presentes em qualquer implementação do padrão CORBA (*Delegate* e POA).

6.2 Integração do UMIOP ao *JacORB*

Para implementação das especificações UMIOP, escolheu-se o *JacORB*, um ORB não comercial e de código aberto, disponibilizado pela *Freie Universität Berlin*. Este ORB foi escolhido devido a sua reconhecida qualidade e desempenho e pela experiência do grupo de sistemas distribuídos do LCMI/DAS com o mesmo nos projetos GROUPPAC [31, 32] e JACOWEB [49].

Antes de apresentarmos a implementação do UMIOP no *JacORB* apresentaremos a arquitetura deste, explicitando como ocorre o processamento de uma requisição tanto no cliente quanto no servidor.

6.2.1 Arquitetura do *JacORB*

A figura 6.2 apresenta um diagrama de classes em UML (*Unified Modeling Language*) [5] com as principais classes participantes do processamento de requisições no cliente *JacORB*. As classes apresentadas neste diagrama interagem desde a chamada do método no *stub* por uma aplicação cliente até o envio da mensagem GIOP correspondente através da conexão TCP. O *stub* é o responsável pela serialização da chamada do método, deixando o envio da mensagem a cargo da classe `Delegate`. Esta classe mantém uma conexão TCP aberta para o ORB servidor, onde a implementação correspondente ao *stub* está ativada.

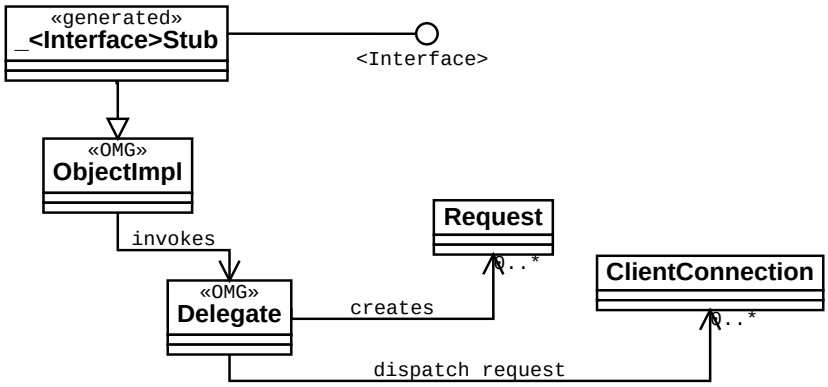

Figura 6.2: Processamento de requisições no cliente *JacORB*.

No lado do servidor, o processamento de requisições é bem mais complexo. A figura 6.3 apresenta um diagrama de classes com as principais participantes do processamento da requisição desde sua chegada pelo *socket* até sua execução pela implementação apropriada. Um objeto da classe `Listener` escuta uma porta de rede a espera de requisições via TCP/IP. Quando uma requisição é recebida nesta porta, o objeto cria uma nova instância da classe `RequestReceptor`, que é a responsável por interpretar a requisição e encaminhá-la ao POA destino. Uma vez no POA, a requisição é colocada em uma fila de processamento para execução pela implementação destino (subclasse do *skeleton* que estende o `Servant`). Note que várias classes presentes na figura 6.3 têm o estereótipo *thread*, estas classes representam os pontos da estrutura do *JacORB* que implementam sua arquitetura *multithreading*.

Nestes dois diagramas, as classes com o estereótipo "OMG"representam implementações de interfaces que fazem parte do padrão CORBA, e portanto estão presentes em qualquer implementação desta arquitetura.

6.2.2 Arquitetura do Módulo UMIOP

Para o processamento de mensagens MIOP, uma nova estrutura de classes teve de ser implementada e adicionada àquelas presentes nas figuras 6.2 e 6.3. A figura 6.4 apresenta um diagrama com esta estrutura.

Neste diagrama podemos identificar o ponto do processamento de requisições do ORB que é alterado de modo que mensagens também possam ser enviadas por difusão: a classe `Delegate`. É nesta classe que, quando a referência do cliente é uma referência de grupo e a mensagem é *oneway* (sem resposta), o processamento é desviado para a classe `MulticastSender`, que fragmenta a requisição GIOP e encapsula estes fragmentos em pacotes MIOP que são difundidos via *multicast* IP. No servidor, um objeto da classe `MulticastListener` é criado para cada grupo em que o ORB tem membros registrados (um *listener* para cada porta e uma porta para cada grupo) pelo adaptador

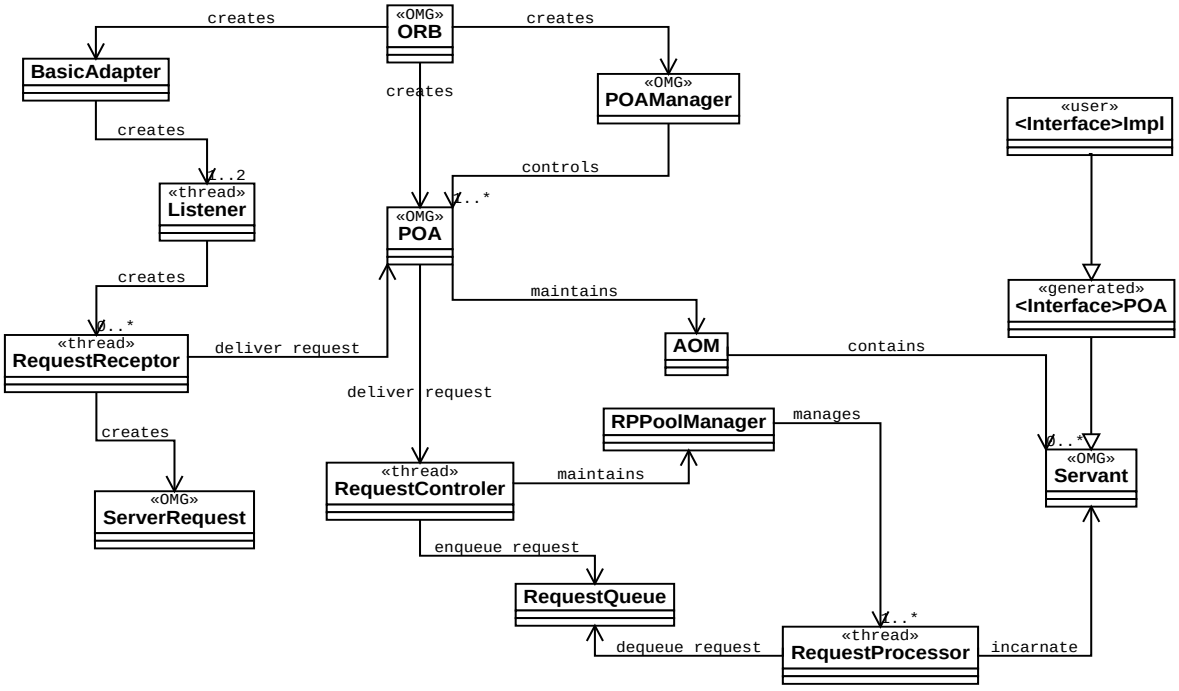

Figura 6.3: Processamento de requisições no servidor *JacORB*.

multicast (classe *MulticastAdapter*). Estes *listeners* recebem os pacotes MIOP e armazenam os fragmentos da mensagem encapsulada neles. Quando a mensagem está completa, uma *thread* (classe `MulticastRequestReceptor`) é retirada do *pool* de *threads* (classe `MRRFactory`) e disparada para unificação dos fragmentos da mensagem GIOP original (que correspondem a coleção de pacotes MIOP) que é então repassada a todos os POAs do ORB. Em cada POA o mapa de grupos ativos (classe `AGM`), relaciona as informações do grupo contidas no cabeçalho da mensagem GIOP com o conjunto de identificadores de objetos pertencentes a este grupo que foram registrados neste POA.

Note que a requisição é encaminhada a todos os POAs ativos no ORB, mesmo aqueles que não registram implementações de objetos pertencentes ao grupo destino, este algoritmo de entrega foi desenvolvido visando tornar o AGM o mais simples possível e evitar o *overhead* de processamento imposto pela busca de POAs destino. Outro fator que justifica esta implementação é o fato de que na maioria das aplicações, não se ativa um grande número de POAs.

6.3 Implementação ReMIOP

Esta seção apresenta o modelo de integração do ReMIOP ao MJACO. Será definido primeiramente o conceito de estratégia de confiabilidade plugável e depois será descrita a implementação do ReMIOP propriamente dita.

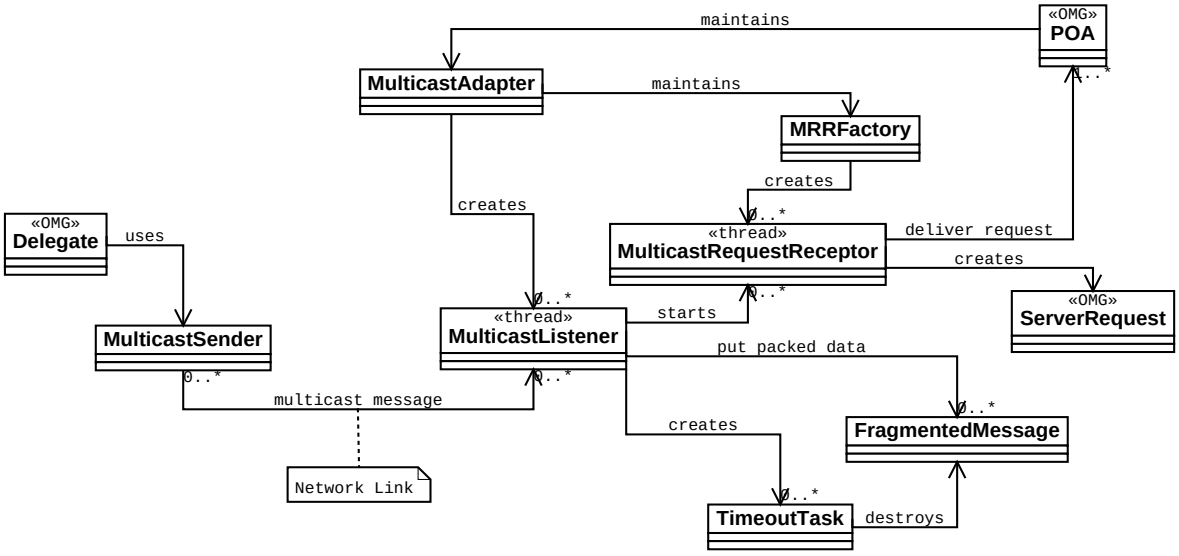

Figura 6.4: Extensões do MJACO.

6.3.1 Estratégias de Confiabilidade

Uma vez definido o protocolo de difusão confiável a ser utilizado integrado ao MIOP, o ReMIOP, este foi implementado no MJACO. Tendo em vista as considerações a respeito do espaço de projeto para protocolos de difusão confiável [24, 50], que sugere protocolos diferentes para aplicações diferentes, e as limitações do ReMIOP, optou-se por acrescentar ao MJACO a capacidade de integrar protocolos plugáveis ao protocolo MIOP através da implementação do padrão de projeto Estratégia (*Strategy*) [19], de uma forma tal que a estratégia de confiabilidade do MIOP possa ser trocada a qualquer momento sem alterações na implementação do ORB.

O padrão estratégia consiste na separação de um algoritmo de seu executor (objeto) e em seu encapsulamento em outra classe. Isto permite que o comportamento do objeto, o algoritmo, possa ser trocado a qualquer momento. A principal vantagem da utilização deste padrão é a capacidade de trocar o comportamento de um objeto sem alterar seu código, trocando apenas a estratégia utilizada por este. No caso do MJACO, trocamos o protocolo confiável integrado ao MIOP sem alterar nossa implementação do padrão UMIOP.

A figura 6.5 ilustra a implementação deste padrão no MJACO.

Nesta figura, temos a classe `MulticastUtil`, uma classe que oferece vários valores de parâmetros para a implementação do MIOP no MJACO, um destes parâmetros é a estratégia de implementação de confiabilidade, representada pela interface `ReliabilityStrategy`. Esta interface define 7 métodos, que são chamados pelo ORB:

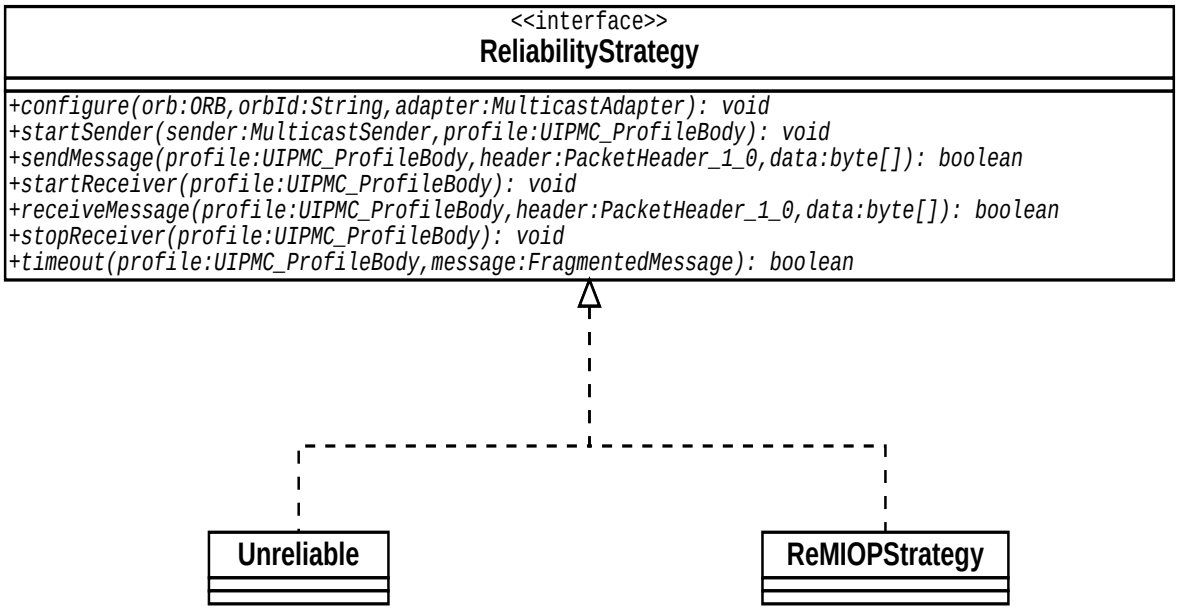

Figura 6.5: Aplicação do padrão de projeto *Strategy* no MJACO.

- **`configure`:** Método chamado quando o ORB é inicializado. Este método recebe como parâmetros uma série de referências e informações úteis para a estratégia de confiabilidade, como por exemplo o ORB, seu identificador e uma referência ao adaptador *multicast*;
- **`startSender`:** Este método é chamado na primeira vez em que o ORB tenta difundir uma mensagem em um determinado grupo. As implementações da estratégia geralmente realizam a alocação de recursos para um emissor neste método;
- **`sendMessage`:** Este método é invocado sempre que o ORB tenta difundir um pacote MIOP. Se retornar `true` o ORB segue processando e difunde a mensagem normalmente, caso contrário a mensagem não é difundida pelo módulo UMIOP (mas pode ser difundida pela estratégia);
- **`startReceiver`:** Método invocado quando o ORB cria um `MulticastListener` para o recebimento de mensagens referentes a um grupo. A chamada deste método indica que pelo menos um membro do grupo em questão está ativo no ORB, o que o caracteriza como um receptor MIOP, assim a implementação deste método caracteriza-se como o local propício para a alocação de recursos referentes a um receptor;
- **`receiveMessage`:** Este método é chamada toda vez que o ORB recebe um pacote MIOP. Se retornar `true` o ORB segue processando o pacote e o entrega aos POAs, caso contrário a mensagem é deixada de lado;
- **`stopReceiver`:** Método invocado quando não existem mais objetos pertencentes a este grupo mantidos pelo ORB. Este método é o local propício para a liberação de recursos utilizados pelos receptores, como por exemplo *buffers*;

- **`timeout`:** Método invocado antes de uma coleção de pacotes MIOP incompleta ser descartada. A implementação deste método permite que a estratégia de confiabilidade tome algum tipo de providência para recuperar o(s) pacote(s) ausente(s). Se este método retornar `true` a coleção é descartada, caso contrário ela permanece na memória.

Até o momento, foram realizadas duas implementações de estratégias de confiabilidade para o MJACO:

- **`Unreliable`:** Classe vazia que apenas autoriza a continuação do processamento das mensagens pelo módulo UMIOP. Esta classe existe para permitir a utilização de MIOP puro e é utilizada como estratégia de confiabilidade padrão;
- **`ReMIOPStrategy`:** Esta classe implementa o protocolo ReMIOP. Ela implementa os algoritmos especificados no capítulo 5 através da concretização da interface `ReliabilityStrategy` (ver próxima seção).

Apesar de atualmente estarem implementadas apenas duas estratégias, esse *framework* permite a integração qualquer protocolo de comunicação de grupo ao MIOP bastando apenas que esse seja concebido como uma implementação da interface `ReliabilityStrategy` e que o nome desta implementação seja definido no arquivo de propriedades do MJACO.

6.3.2 Estratégia ReMIOP

A estratégia ReMIOP foi implementada através de um mapeamento das abstrações utilizadas neste protocolo como mensagem, grupo e *buffer* em classes Java acionadas a partir de uma implementação da interface `ReliabilityStrategy` chamada `ReMIOPStrategy`. O diagrama de classes UML que representa esta implementação é apresentado na figura 6.6.

No diagrama da figura 6.6 a classe `ReMIOPStrategy` é a base para toda a implementação do ReMIOP. Esta classe mantém uma lista dos grupos que se comunicam com este ORB. Cada grupo é representado pela classe `Group`, classe esta que mantém um conjunto de *buffers* (classe `SortedSet`) gerenciados pela classe `BufferManager`. Cada um destes *buffers* é associado a um emissor do grupo. Se o ORB trabalha como emissor também ele instancia um objeto da classe `ToSendList` para armazenar as mensagens enviadas e a enviar.

As mensagens são sempre enviadas pela classe `Group` através da tarefa `MulticastTask` (dados ou NACK) ou pela tarefa `StateTask` (mensagem de estado). Estas tarefas são registradas em um escalonador (classe `java.util.Timer`).

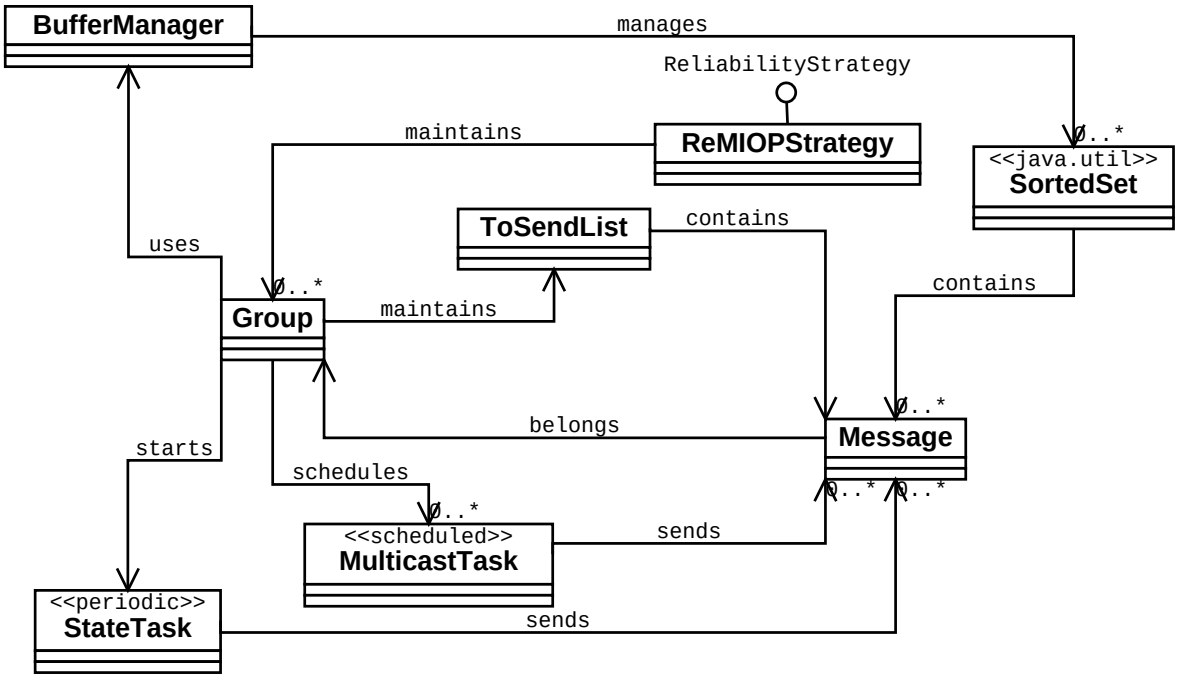

Figura 6.6: Implementação da estratégia ReMIOP.

6.4 Testes de Desempenho e Escalabilidade

Nesta seção são apresentamos alguns testes simples realizados com o objetivo de validar e verificar o desempenho da implementação do MJACO. Os testes foram executados no LCMI/DAS usando uma rede local e 4 computadores com configurações semelhantes[1], conectados através de um mesmo *hub*. Foram implementadas duas versões de um programa distribuído que mede o tempo de *round-trip*[2]: uma utilizando *sockets multicast* e outra fazendo uso do MJACO.

O primeiro experimento foi montado da seguinte forma: duas instâncias do programa foram iniciadas em cada uma das quatro maquinas, totalizando oito membros em um mesmo grupo. Um destes membros é o transmissor que envia uma mensagem de tamanho variável para o grupo e que, posteriormente, fica aguardando o recebimento de todas as confirmações dos membros receptores do grupo. Esse processo é repetido por 10000 vezes. O experimento foi executado com cada uma das versões do programa exemplo e os resultados obtidos são apresentados na figura 6.7.

No gráfico da figura 6.7 é possível verificar que a utilização de *sockets multicast* resulta em um desempenho aproximadamente 60% melhor (em média) do que a utilização do MJACO. O que já era

[1]Pentium III 900MHz com 198 MB de memória RAM e utilizando interfaces de rede Ethernet de 10Mbps. O sistema operacional utilizado é o Suse Linux 7.0.

[2]Tempo necessário para a difusão de uma mensagem em um grupo e o recebimento da resposta de todos os seus membros.

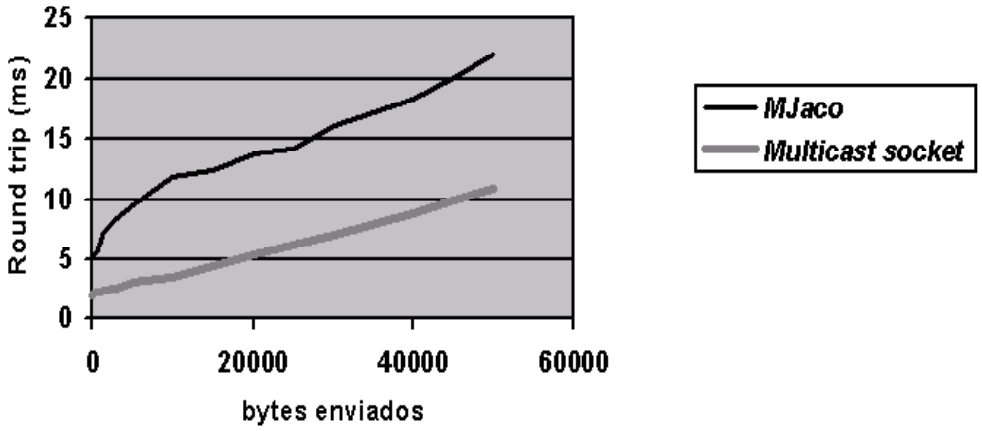

Figura 6.7: Desempenho do MJACO.

esperado, visto que o MJACO coloca toda uma camada de software para gerenciamento de objetos e requisições bem como para a serialização das mensagens (ver figura 6.1); estrutura essa que possivelmente as aplicações que utilizam-se de *sockets multicast* teriam de implementar de alguma forma, e que o nosso pequeno programa de teste não implementa. Vale ressaltar também que a implementação do MJACO foi realizada o menos dependente possível do *JacORB*, o que acarreta algumas perdas de desempenho, que devem ser minimizadas conforme estes dois projetos forem se integrando e as implementações forem evoluindo.

Um ponto importante a se enfatizar é que, na configuração apresentada, a partir das mensagens de 50000 bytes, o sistema começa a perder mensagens. Este comportamento se verificou tanto na versão com *sockets multicast* quanto na que utiliza o MJACO, e deve-se a não confiabilidade do UDP, que não possui mecanismos de controle de fluxo para evitar os descartes de pacotes decorrentes de *buffers* cheios, durante o alto congestionamento da rede local nos testes. A implementação deste teste utilizando a estratégia ReMIOP acoplada ao MJACO com parâmetros de configuração do protocolo, em especial as taxa de envio máxima e miníma (R_{min} e R_{max}), definidos adequadamente acabou com este problema já que o ajuste da taxa de envio tem influência direta sobre a intensidade das difusões, que reflete na pouca ocorrência de descartes de pacotes e no desempenho das comunicações.

O segundo experimento realizado visa avaliar a escalabilidade do MJACO. Os testes foram realizados da seguinte forma: em cada *host* foi iniciada uma instância do programa teste especificando-se quantos objetos do grupo de testes deveriam ser instanciados e registrados nela. Um dos vários objetos criados nos *hosts* do sistema é o transmissor, ou seja, é ele que transmite a mensagem com 4Kbytes de dados e que faz a medição do tempo de *round-trip*. O número de mensagens difundidas durante um *round-trip* é $4n+1$ onde n é o número de objetos pertencentes ao grupo registrados em cada *host* (uma mensagem e um reconhecimento para cada objeto membro do grupo).

O processo de *round-trip* foi repetido para um número variado de objetos por *host* e os resultados são mostrados na figura 6.8.

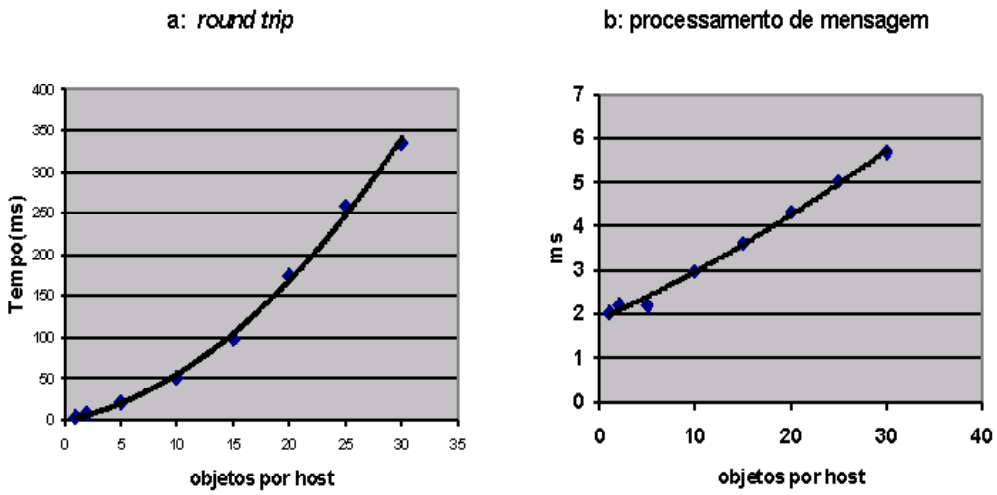

Figura 6.8: Escalabilidade do MJACO.

O gráfico da figura 6.8 mostra a situação óbvia em que, a medida que adicionamos membros no sistema, o tempo de *round-trip* aumenta bastante. Isto se deve claramente ao aumento substancial na quantidade de mensagens geradas na rede, ocupando o meio físico. Por exemplo, com 2 objetos por *host* (8 objetos membros no grupo), temos 9 mensagens trocadas (1 mensagem de dados e 8 de reconhecimentos) para um *round-trip*. Para 10 objetos por *host* temos 41 mensagens trocadas. Levando em conta o tempo de duração de um *round-trip* e dividindo o mesmo pelo seu número de mensagens, é obtida uma estimativa de tempo médio para transferência de uma mensagem no sistema. O gráfico "b"da figura 6.8 apresenta estes valores em função das composições dos grupos. Neste gráfico é possível notar que a curva que representa este tempo médio das mensagens é linear, e aumenta aproximadamente 0.1 ms para cada novo objeto incluído em cada *host* do sistema, o que demonstra o potencialidade para escalabilidade do MJACO.

Os testes apresentados foram feitos utilizando a estratégia *Unreliable* acoplado ao MJACO. Testes com a estratégia ReMIOP não foram realizados pois o desempenho do ORB é por demais dependente dos parâmetros definidos para o ReMIOP (R_{min}, R_{max}, P_{nack}, P_{state} e P_{repair}), o que dificulta muito uma análise quantitativa deste protocolo, entretanto algumas considerações podem ser feitas:

- **Quanto ao Desempenho:** Temos como fator crítico de nossa implementação, em termos de desempenho, o algoritmo 6, que procura lacunas em *buffers* e agenda NACKs. Este algoritmo utiliza um iterador fornecido pela classe `SortedSet`, para procurar linearmente no conjunto de mensagens, assim, o algoritmo pode ser classificado como $O(n)$ com n sendo o tamanho máximo do *buffer*, o que garante um bom desempenho para *buffers* não muito grandes;

- **Quanto à Escalabilidade:** Se considerarmos um grupo para o qual existem *n* emissores, e onde os *buffers* tem capacidade para *m* mensagens, teremos *nm* mensagens armazenadas em cada receptor deste grupo. Logo a quantidade de memória utilizada nos receptores é linearmente proporcional ao **número de emissores** no grupo, o que caracteriza independência do tamanho do grupo.

6.5 Conclusão

Este capítulo apresentou os principais resultados referentes à implementação do MIOP e do ReMIOP no *JacORB*. Estas implementações culminaram na criação do ORB MJACO. Os passos percorridos para a implementação deste ORB foram definidos a partir do modelo de integração proposto neste trabalho e do padrão de projeto estratégia.

Os testes realizados nesta implementação apontam custos de desempenho não favoráveis ao MJACO em relação aos *sockets multicast*, pois há uma perda de eficiência, que é compensada, se levarmos em consideração as vantagens da programação de objetos distribuídos. O gerenciamento de objetos e *threads*, totalmente a cargo do POA e do ORB, e a interoperabilidade fornecida pelo CORBA adicionam simplicidade a aplicações que antes se limitavam a *sockets* UDP e outras interfaces de baixo nível.

Capítulo 7

Considerações Finais e Perspectivas Futuras

A abstração de grupo de objetos é um modelo adequado para o tratamento de serviços tolerantes a faltas em sistemas distribuídos. Estes serviços são implementados através de grupos de objetos replicados que podem fornecer o serviço independentemente de falhas em algum membro do grupo. Toda implementação de grupo de objetos em sistemas distribuídos necessita de um suporte de comunicação de grupo. O *Unreliable Multicast Inter-ORB Protocol* (UMIOP) é o primeiro passo para outras soluções de comunicação de grupo na arquitetura CORBA, como serviços com garantias mais fortes como as fornecidas pelo ReMIOP.

Este trabalho apresentou um estudo sobre a integração de serviços de difusão em *middleware* CORBA. Inicialmente trabalhamos com difusão não confiável, seguindo os padrões da OMG, a fim de se implementar uma base para serviços com requisitos de garantia mais fortes. Como um segundo passo neste sentido, introduzimos o ReMIOP, com uma semântica de comunicação confiável. O ReMIOP define um conjunto de extensões ao MIOP, em particular controle de fluxo e retransmissões de pacotes, de forma que este possa ser utilizado como suporte de comunicação de grupo para implementações do padrão FT-CORBA.

O modelo de integração proposto para integração do UMIOP não compromete aspectos de interoperabilidade e portabilidade do ORB como um todo. O ORB é capaz de realizar invocações tanto usando o IIOP (convencional) como o MIOP, e ainda o ReMIOP. Este modelo pode muito bem ser adotado para integrar outros protocolos de comunicação. A implementação realizada deste modelo deu origem ao ORB MJACO, que estende o *JacORB* (um ORB Java de código aberto) para fornecer todos estes serviços de comunicação de grupo, e ainda suporte a múltiplos protocolos de difusão confiável, através da implementação do padrão de projeto *Strategy*. Estas implementações podem ser obtidas na WEB no seguinte endereço (*http://grouppac.sourceforge.net/*)

O modelo de integração entre o UMIOP (com as extensões ReMIOP) e o FT-CORBA será aproveitado no projeto GROUPPAC, que implementa a especificação *Fault-Tolerant* CORBA, no sentido da concepção de modelos de replicação ativa, inexistentes nas atuais especificações da OMG. A própria OMG já está reunindo um grupo para trabalhar em uma RFP sobre difusão confiável no CORBA. Estes serviços de difusão com garantias devem ser implementados sobre o MIOP, nos mesmos moldes do ReMIOP.

Entre os possíveis trabalhos futuros podemos ressaltar:

- Implementação do modelo de integração do UMIOP com o FT-CORBA através da junção das implementações do GROUPPAC e do MJACO;
- Verificação do ReMIOP em modelos formais de sistemas distribuídos parcialmente síncronos, como o assíncrono temporizado [12] e o quase síncrono [48], aproveitando-se da definição formal dos algoritmos deste apresentada neste trabalho;
- Estudo e extensão do ReMIOP afim de torná-lo adaptável a redes de comportamento inconstante, utilizando para isso mecanismos adaptativos para ajuste dos parâmetros do protocolo. Dentre estes mecanismos podemos citar redes neurais, algoritmos genéticos e sistemas nebulosos;
- Ainda no âmbito de sistemas adaptativos, utilizar a estrutura de protocolos plugáveis do MJACO, os componentes para a concepção de protocolos de difusão confiável definidos pelo IETF e os recentes esforços da OMG para definição de uma interface para protocolos plugáveis padronizada para definir um *middleware* que troque automaticamente de protocolo de comunicação de acordo com a aplicação e o ambiente em que ele está sendo usado;

Estes trabalhos estarão sendo desenvolvidos pelo grupo de sistemas distribuídos do DAS e pelo autor como trabalho de doutorado, se aproveitando da base de software do MJACO e do GROUPPAC.

Referências Bibliográficas

[1] BARROS, F. A. R., AND STANTON, M. Modelo de serviço de difusão ip: Aperfeiçoamentos ou mudanças. In *Anais do 19o. Simpósio Brasileiro de Redes de Computadores* (Floarianópolis - SC, 2001).

[2] BESSANI, A. N., DA SILVA FRAGA, J., AND LUNG, L. C. Mjaco - integração do multicast ip na arquitetura corba. In *Anais do 20o. Simpósio Brasileiro de Redes de Computadores* (Buzios - RJ, 2002).

[3] BIRMAN, K. P. *Building Secure and Reliable Network Applications*. Manning, Greenwich, 1996.

[4] BIRMAN, K. P., HAYDEN, M., OZKASAP, O., XIAO, Z., BUDIU, M., AND MINSKY, Y. Bimodal multicast. *ACM Transactions on Computer Systems 17*, 2 (May 1999), 41–88.

[5] BOOCH, G., JACOBSON, I., RUMBAUGH, J., AND RUMBAUGH, J. *The Unified Modeling Language User Guide*. Addison-Wesley, Reading, 1998.

[6] BROSE, G. Jacorb: Implementation and design of a javaorb. In *Proceedings of IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems* (1997).

[7] CHANDRA, T. D., HADZILACOS, V., AND TOUEG, S. The weakest failure detectors for solving consensus. *Journal of the ACM 43*, 4 (July 1996).

[8] CHANDRA, T. D., HADZILACOS, V., TOUEG, S., AND CHARRON-BOST, B. On the impossibility of group membership. In *Proceedings of Fifteenth ACM Symposium on Principles of Distributed Computing* (1996), pp. 322–330.

[9] CHANDRA, T. D., AND TOUEG, S. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM 43*, 2 (Mar. 1996).

[10] CHANG, J., AND MAXEMCHUK, N. Reliable broadcast protocol. *ACM Transactions on Computer Systems 2*, 3 (Aug. 1984), 251–273.

[11] CHIU, D.-M., HURST, S., KADANSKY, M., AND WESLEY, J. Tram: A tree-based reliable multicast protocol. Tech. rep., Sun Microsystems Laboratories, Sun Microsystems, California - USA, July 1998.

[12] CRISTIAN, F., AND FETZER, C. The timed asynchronous distributed system model. *IEEE Transactions on Parallel and Distributed Systems 10*, 6 (June 1999), 642–657.

[13] DEERING, S. E. Host extensions for ip multicasting (rfc 988). IETF Request For Comments, July 1986.

[14] DEERING, S. E., AND CHERITON, D. R. Host groups: A multicast extension to the internet protocol (rfc 966). IETF Request For Comments, Dec. 1985.

[15] DELPORTE-GALLET, C., FAUCONNIER, H., AND GUERRAOUI, R. A realistic look at failure detectors. In *Proceedings of the IEEE Symposium on Dependable Systems and Networks (DSN 2002)* (Washington DC, 2002).

[16] DOLEV, D., AND MALKI, D. The transis approach to high avaliability cluster communication. *Communications of the ACM 39*, 4 (Apr. 1996), 64–70.

[17] FISCHER, M. J., LYNCH, N. A., AND PATERSON, M. S. Impossibility of distributed consensus with one faulty process. *Journal of the ACM 32*, 2 (Apr. 1985), 374–382.

[18] FLOYD, S., JACOBSON, V., LIU, C.-G., MCCANE, S., AND ZHANG, L. A reliable multicast framework for light-weight session and application level framing. *IEEE/ACM Transactions on Networking* (Dec. 1997).

[19] GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, 1995.

[20] GROUP, O. M. Fault-tolerant corba specifi cation v1.0. OMG Standart, 2000.

[21] GROUP, O. M. The common object request broker architecture specifi cation v2.6. OMG Standart, 2001.

[22] GROUP, O. M. Unreliable multicast inter-orb protocol specifi cation. OMG Standart, 2001.

[23] HADZILACOS, V., AND TOUEG, S. A modular approach to the specifi cation and implementation of fault-tolerant broadcasts. Tech. rep., Department of Computer Science, Cornell University, New York - USA, May 1994.

[24] HANDLEY, M., FLOYD, S., WHETTEN, B., KERMODE, R., VICISANO, L., AND LUBY, M. The reliable multicast design for bulk data transfer (rfc 2887). IETF Request For Comments, Aug. 2000.

[25] HANNA, S., KADANSKY, M., AND ROSENZWEIG, P. Java reliable multicast service overview. Tech. rep., Sun Microsystems Laboratories, Sun Microsystems, California - USA, Sept. 1998.

[26] KUMAR, V. *MBone: Interactive Multimedia on the Internet*. New Riders, Indianapolis, 1995.

[27] LAMPORT, L. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM 21*, 7 (July 1978), 558–565.

[28] LEVINE, B. N., AND GARCIA-LUNA-ACEVES, J. A comparison of reliable multicast protocols. *Multimedia Systems - Springer Verlag*, 6 (1998), 334–348.

[29] LIAO, T. Light-weight reliable multicast protocol. Disponível em http://webcanal.inria.fr/lrmp/, 1998.

[30] LIN, J. C., AND PAUL, S. Rmtp: A reliable multicast transport protocol. In *Proceedings of IEEE INFOCOM'96* (1996), pp. 1414–1424.

[31] LUNG, L. C. *Experiências com Tolerância a Faltas no CORBA e Extensões no FT-CORBA para Sistemas Distribuídos de Larga Escala*. Tese de doutorado em engenharia elétrica, Centro Tecnológico, Universidade Federal de Santa Catarina, Florianópolis, 1996.

[32] LUNG, L. C., DA SILVA FRAGA, J., FARINES, J. M., AND OLIVEIRA, J. R. Experiências com comunicação de grupo nas especifi cações fault tolerant corba. In *Anais do 18o. Simpósio Brasileiro de Redes de Computadores* (Belo Horizonte - MG, 2000).

[33] MANKIN, A., ROMANOW, A., BRADNER, S., AND PAXSON, V. The ietf criteria for evaluating reliable multicast transport and application protocols (rfc 2357). IETF Request For Comments, June 1998.

[34] MELLIAR-SMITH, P., MOSER, L., AND AGRAWALA, V. Broadcast protocols for distributed systems. *IEEE Transactions on Parallel and Distributed Systems 1*, 1 (Jan. 1990).

[35] MISHRA, S., PETERSON, L. L., AND SCHLICHTING, R. D. Consul: A communication substrate for fault-tolerant distributed programs. *Distributed Systems Engineering Journal 1*, 2 (Dec. 1993), 87–103.

[36] MONTRESOR, A. The jgroup reliable distributed object model. Tech. rep., Department of Computer Science, University of Bologna, Bologna - Italy, Mar. 1999.

[37] MOSER, L., MELLIAR-SMITH, P., NARASIMHAN, P., KOCH, R., AND BERKET, K. A multicast group communication protocol, engine, and bridge for corba. *Concurrency and Computation Pratice and Experience 13*, 7 (June 2001), 579–603.

[38] MOSER, L. E., MELLIAR-SMITH, P. M., AGARWAL, D. A., BUDHIA, R. K., AND LINGLEY-PAPADOPOULOS, C. A. Totem: A fault-tolerant multicast group comunication system. *Communications of the ACM 39*, 4 (Apr. 1996), 54–63.

[39] POSTEL, J. User datagram protocol (rfc 768). IETF Request For Comments, Aug. 1980.

[40] POWEL, D. Group communication. *Communications of the ACM 39*, 4 (Apr. 1996), 50–53.

[41] RODRIGUES, L., AND VERÍSSIMO, P. xamp: a multi-primitive group communications service. In *Proceedings of the 11th Symposium on Reliable Distributed Systems* (Houston - Texas, 1992).

[42] SABATA, B., BROWN, M., DENNY, B., AND HEO, C. H. Transport protocol for reliable multicast: Trm. In *Proceedings of the International Conference on Networks* (Orlando - Flórida, 1996).

[43] SCHNEIDER, F. B. Implementing fault-tolerant service using the state machine aproach: A tutorial. *ACM Computing Surveys 22*, 4 (Dec. 1990), 299–319.

[44] SIEGEL, J. Omg overview: Corba and the oma in enterprise computing. *Communications of the ACM 41*, 10 (Oct. 1998).

[45] STRAYER, T., DAMPSEY, B., AND WEAVER, A. *XTP: The Xpress Transfer Protocol*. Addison-Wesley, Reading, 1992.

[46] VAN RENESSE, R., BIRMAN, K. P., AND MAFFEIS, S. Horus: A flexible group communication system. *Communications of the ACM 39*, 4 (Apr. 1996), 76–83.

[47] VAN RENESSE, R., BIRMAN, K. P., MARK HAYDEN, A. V., AND KARR, D. Building adaptative systems using ensemble. *Software - Pratice and Experience 28*, 9 (Aug. 1998), 963–979.

[48] VERISSIMO, P., AND ALMEIDA, C. Quasi-synchronism: A step away from the traditional fault-tolerant real-time system models. *IEEE TCOS Bulletin 7*, 4 (Dec. 1995).

[49] WANGHAN, M. S., LUNG, L. C., WESTPHALL, C. M., AND FRAGA, J. S. Integrating ssl to jacoweb security framework: Project and implementation. In *Proceedings of IEEE/IFIP International Symposium on Integrated Network Management* (2001), pp. 779–792.

[50] WHETTEN, B., VICISANO, L., KERMODE, R., HANDLEY, M., FLOYD, S., AND LUBY, M. Reliable multicast transport building blocks for one-to-many bulk-data transfer (rfc 3048). IETF Request For Comments, Jan. 2001.

[51] YAVATKAR, R., GRIFFIOEN, J., AND SUDDAN, M. A reliable dissemination protocol for interactive collaborative applications. In *Proceedings of ACM Multimedia* (1995), pp. 333–344.